HYBRID VIDEO CODING DESIGN WITH VARIABLE SIZE INTEGER TRANSFORMS AND STRUCTURAL SIMILARITY

by

ATT KRUAFAK

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2008

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. K. R. Rao, my supervisor, for his kind guidance and support throughout the time of my research and graduate studies. His tireless dedication to his students and maintaining dynamic research community has inspired my work.

I also would like to express my gratefulness to Dr. Michael T. Manry, Dr. Wei-Jen Lee, Dr. Wendell Alan Davis, and Dr. Kambiz Alavi for their guidance as members of my graduate committee. I truly thank Dr. Zhou Wang who was a committee member of my comprehensive exam for his valuable guidance. I sincerely thank Dr. Zhang Zheng Bing, Dr. Jung-ho Lee, Dr. Do Nyeon Kim, and Dr. Tokunbo Ogunfunmi for attending my defense and comprehensive exams.

I appreciate all members of Multimedia Processing Lab for their valuable assistances, discussions and friendship. This includes, Dr. Xuejun Hu (EE'05) and Dr. Lin Tong (EE'05) for initial source code starting my video simulation, Dr. Do Nyeon Kim for his suggestion on integer transforms and motion prediction discussion, Qiang Li (EE) for SSIM discussion. I also would like to thank CAT Telecom public company for financial support of my Ph.D. study.

I cordially thank Thai colleagues at UT Arlington for their friendship and support, including, Dr. Chai Chompoo-inwai (EE'05) for sharing the beginning of Ph.D. study life and experience, Dr. Yodchanan Wongsawat (EE'07) for strategic comments on my research, Chivalai Themiyasathit (IE) and Dr. Wikrom Prombutr (Business'08) for their questions on their research programming that technically turn to improving programming skills in my research.

Finally, I am grateful to three most important persons; my mother, Ms. Poonsiri Kruafak, my father, Mr. Prajuab Kruafak, and my younger brother, Tee Kruafak, pharmacist (his sore throat and cold medicine) for their innumerable support and encouragement during my Ph.D. journey.

October 29, 2008

ABSTRACT

HYBRID VIDEO CODING DESIGN WITH VARIABLE SIZE INTEGER TRANSFORMS AND STRUCTURAL SIMILARITY

Att Kruafak, PhD.

The University of Texas at Arlington, 2008

Supervising Professor: K.R. Rao

This research proposes a block-based video codec design with two objectives. The first goal is to propose a method for intraframe that improves the rate-distortion (peak signal-to-noise ratio versus bit rate) of a fixed-size transform encoder. The proposed method uses three integer transform sizes (4×4), (8×8), and (16×16). The codec also adopts H.264-like spatial prediction to intraframe encoding. For simplicity of the design, Huffman variable-length code is used as entropy encoding. For intraframe encoding, the simulations show rate-distortion improvement over JPEG and JPEG2000. In some test sequences, the simulations also show improvement over H.264 (baseline profile at low complexity mode without rate-distortion optimization) with a small increase of operations on each macroblock at the decoder side.

The second goal of this research is to study rate-distortion behavior of the interframe codec with novel motion estimation based on structural similarity (SSIM) and the codec with conventional motion estimation based on pixel error distortion (sum of absolute difference). A

study from previous literature shows that the structural similarity metric provides better image assessment than a pixel error based metric (mean square error and peak signal-to-noise ratio). Structural similarity measurement on the true color components (RGB) with equal weight for each component is proposed. The results on rate-distortion show that both structural similarity and peak signal-to-noise ratio (PSNR) provide similar measurements. Both sum of absolute difference (SAD)- and structural similarity (SSIM)-based distortions in motion prediction of large block sizes, {(16×16), (8×8)}, have similar performances. For the small block size of (4×4), SAD-based distortion provides better rate-distortion performance. Distortion calculation for SSIM requires more operations compared to SAD.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ACRONYMS

ABT	Adaptive block size transform
AVC	Advanced video coding
avg	Average
AVS	Audio video coding standard
BD-PSNR	Bjonteggard delta peak signal-to-noise ratio
bpp	Bits per pixel
bps	Bits per second
BSS	Block size selection
CBFPS	Center-biased fractional pixel search
CIF	Common intermediate format
coeffBSS	Coefficient block size selection
dB	Decibel
DCT-II	Discrete cosine transform type II
DS	Diamond search pattern
EHS	Extended hexagon-based search
fps	Frames per second
FRExt	Fidelity range extensions
GOP	Group of pictures
H.264	International standard on video coding of ITU-T
HEXBS	Hexagon based search
HVS	Human visual system

ICT	Integer cosine transform
inv	Inverse
ITU-T	International telecommunication union-telecommunication standardization sector
JM	Joint model reference software
JPEG	Joint photographic experts group
JVT	Joint video team
MB	Macroblock
ME	Motion estimation
MPEG	Moving picture experts group
MSE	Mean square error
MSSIM	Mean structural similarity index
Mux	Multiplexer
pel	Pixel
PSNR	Peak signal–to–noise ratio
RD	Rate–distortion
RGB	Red, green, and blue components
RLC	Run-length coding
SAD	Sum of absolute difference
SSIM	Structural similarity index
UMHexagonS	Hybrid unsymmetrical-cross multi-hexagon-grid search
VC-1	Video coding standard adopted by Society of Motion Picture and Television Engineers
VCEG	Video coding experts group
VLC	Variable length code
VQEG	Video quality experts group
WMV-9	Windows Media Video 9

CHAPTER 1

INTRODUCTION

1.1 Introduction

The latest video coding standard of the International Telecommunication Union-Telecommunication Standardization Sector (ITU-T), H.264/AVC [1][2][3][4], is designed with efficient integer transforms. It is difficult to make further improvements for rate-distortion on the existing encoder scheme. Although, the standard defines the two integer transform sizes (4×4) and (8×8), H.264 (baseline profile) relies on the former only. This allows the opportunity to improve the coding efficiency by appropriately using the combinations of integer transforms on a square partitioned residue macroblock of a video component frame. The first objective of the research is to introduce this variable block size transform coding with different integer transforms (three sizes) that can improve the rate-distortion performance. Variable size transform coding and block size selection method are based on [5][6] with some modifications. Different from the previous work, this research utilizes two sets of three integer transforms, applying {(16×16), (8×8), (4×4)} transforms on luminance residue macroblocks, and {(8×8), (4×4), (2×2)} transforms on chrominance residue macroblocks of video frames. The block size selection process is simplified to have a smaller header on each encoded macroblock. This reduces the overall bit rate.

This research studies, designs, and implements the block based hybrid video encoder and decoder for a digitized color video sequence (Figure 1.1) in common intermediate format (CIF) (352×288)-pixels (width×height), in YCbCr color format, and 4:2:0 sampling format. Each video frame contains a Y-luminance component, a Cb-blue color difference component, and a

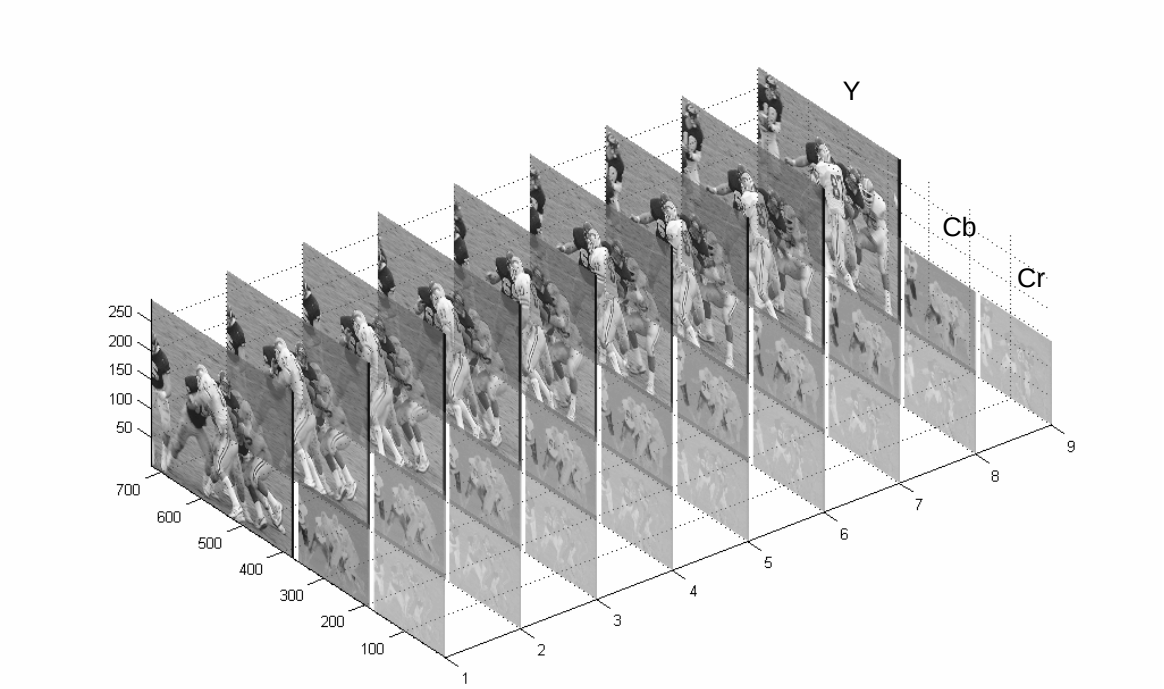


Figure 1.1 Test video sequence, Football, common intermediate format (352×288)-pixels in YCbCr 4:2:0 sampling format, 9 temporal frames. (Show frames 50 to 58.)

Cr-red color difference component, with 8 bits per sample for each component in Figure 1.1.

The proposed hybrid video encoder and decoder are shown in Figures 1.2–1.3. The codec consists of the following encoding elements; H.264-like (16×16)-spatial prediction, integer transforms, H.264-like uniform quantization, run-length encoder (RLC), and Huffman entropy encoder. The codec is simulated in MATLAB. For intraframe encoding, group of pictures (GOP) encoding structure is a sequence of I-frames (III···), where, I is an intra-coded picture. Rate-distortion (as peak signal-to-noise ratio (PSNR) and bit rate) of the codec is compared with JM12.0 [7][8], JPEG2000 (The JasPer project) [9], and JPEG baseline (Independent JPEG group) [10].

In motion estimation of many video coding standards, including H.264/AVC, distortion calculation of block matching is based on a pixel error metric, for example, sum of absolute difference (SAD) and mean square error (MSE). However, the pixel error metric is not designed

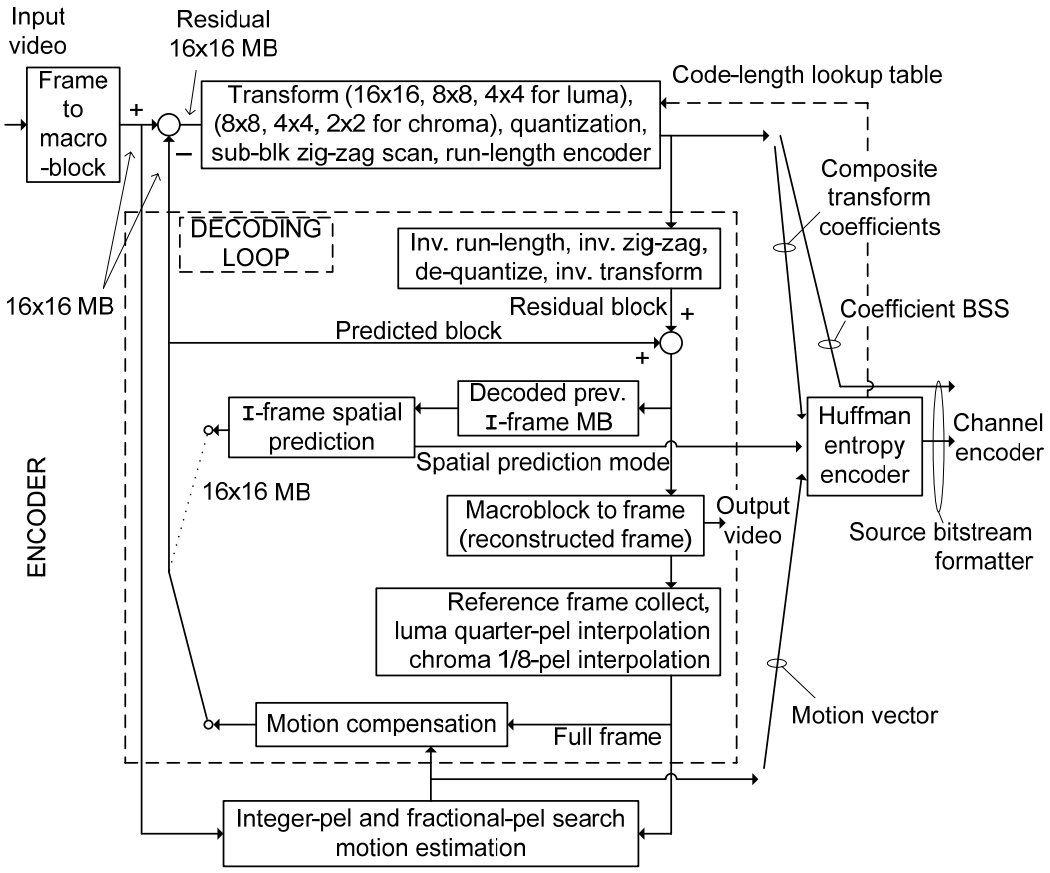


Figure 1.2 The proposed video encoder block diagram. (MB: Macroblock, BSS: Block size selection, pel: pixel)

to match with human visual perception. The disadvantages of the error metric over various types of image degradations can be found in [11][12][13]. Structural similarity (SSIM) index is a new objective quality metric, introduced in [13], for image quality assessment. The second objective of the research is to study and to implement the interframe video coding with the new distortion metric. Video quality assessments based on PSNR and SSIM will be observed. Rate-distortions from two types of distortions, SSIM and SAD (in motion prediction) will be compared.

In the proposed research, the codec is modified with fixed-motion partition size and uses SSIM for block matching distortion calculations to obtain a motion vector during motion search process. MATLAB implementation of SSIM can be obtained from [14]. For interframe coding, the codec uses H.264-like fractional pixel interpolation of reference frame, and motion

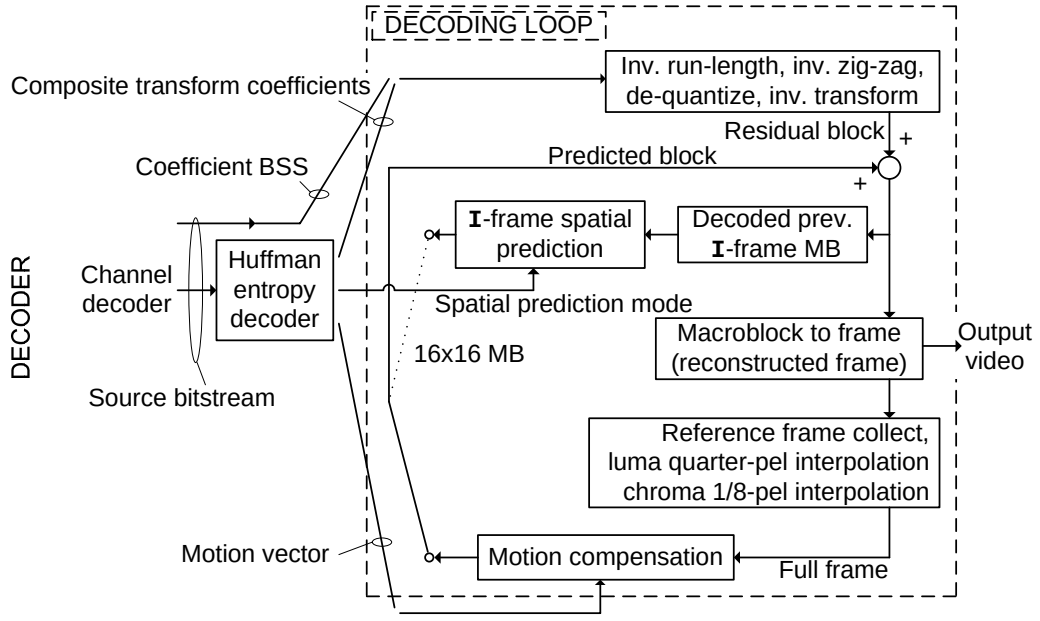


Figure 1.3 The proposed video decoder block diagram

estimation with motion search process used in the fast motion estimation of JM (H.264 reference software) [7][8]. Fixed motion block sizes (16×16), (8×8), and (4×4) are used. Encoded GOP structure is set as IPP···, where, P is (motion) predicted frame. Entropy codebooks are obtained from the training test sequences before the simulation. In the simulation, different reconstructed video qualities in terms of PSNR and SSIM at different bit rates are obtained by varying the quantization parameter. The rate-distortion results from two types of distortions (SAD- and SSIM-based distortions) and two types of codebook training (SAD- and SSIM-based) are compared. Complexities between SAD and SSIM are compared at block level in motion prediction process.

1.2 Background

1.2.1 Block and pixel index calculations

In block based video coding, each frame is partitioned into many equal size blocks, called macroblocks. These macroblocks are processed by block transform coding, block motion search, motion prediction and block reconstruction. This section develops the framework for

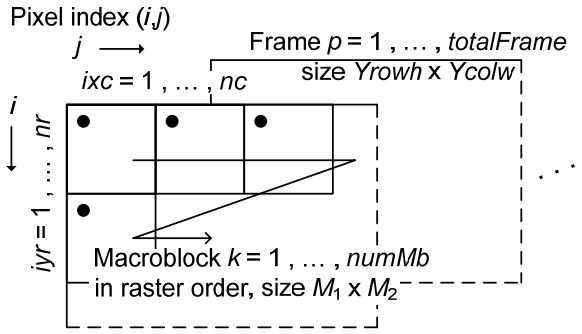


Figure 1.4 Macroblock index numbering and pixel coordinate

block and pixel indexing. Block and pixel indices are used in this codec implementation.

In Figure 1.4, a video frame at size $Yrowh \times Ycolw$ is partitioned into macroblocks of size $M_1 \times M_2$. The macroblock index $(iyr, ixc) \in \{(1,1), \cdots, (nr, nc)\}$ corresponds to the macroblock number, $k \in \{1, \cdots, numMb\}$, where, *iyr* is row index, *ixc* is column index, $nr = Yrowh / M_1$ and $nc = Ycolw / M_2$. The conversion between macroblock index and macroblock number is given by,

$$(iyr, ixc) \to k \;;\quad nc \cdot (ixc - 1) + iyr = k$$

$$k \to (iyr, ixc) \;;\quad \begin{cases} ixc = 1 + \mathrm{mod}(k - 1, nc) \\ iyr = \mathrm{ceil}\left(\dfrac{k}{nc}\right) \end{cases}$$

where, mod is the remainder after the division of $(k - 1)$ by *nc*, and ceil is the largest integer less than or equal to k / nc.

Pixel index (i, j), (*i* is row index or vertical coordinate, and *j* is column index or horizontal coordinate) on each macroblock is calculated by,

$$i_1 : i_{nr} \;;\quad (iyr - 1) \cdot M_1 + 1 : iyr \cdot M_1$$

$$j_1 : j_{nc} \;;\quad (ixc - 1) \cdot M_2 + 1 : ixc \cdot M_2$$

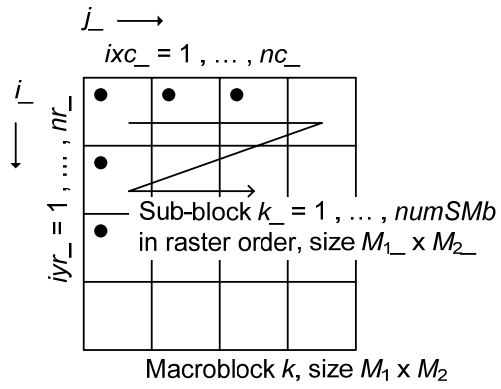


Figure 1.5 Sub-block index numbering and pixel coordinate

For variable block size processes, each macrobock is partitioned into sub-blocks of size $M_{1\_} \times M_{2\_}$ in Figure 1.5. The sub-block index $(iyr\_, ixc\_) \in \{(1,1), \cdots, (nr\_, nc\_)\}$, corresponds to the sub-block number $k\_ \in \{1, \cdots, numSMb\}$, where, $nr\_ = M_1 / M_{1\_}$ and $nc\_ = M_2 / M_{2\_}$. The conversion between sub-block index and sub-block number is given by,

$$(iyr\_, ixc\_) \rightarrow k\_ ; \quad nc\_ \cdot (ixc\_ - 1) + iyr\_ = k\_$$

$$k\_ \rightarrow (iyr\_, ixc\_) ; \quad \begin{cases} ixc\_ = 1 + \mathrm{mod}(k\_ - 1, nc\_) \\ iyr\_ = \mathrm{ceil}\left(\dfrac{k\_}{nc\_}\right) \end{cases}$$

For each sub-block, pixel indices, $(i\_, j\_)$ are calculate by,

$$i_{1\_} : i_{nr\_} ; \quad (iyr\_ - 1) \cdot M_{1\_} + 1 : iyr\_ \cdot M_{1\_}$$

$$j_{1\_} : j_{nc\_} ; \quad (ixc\_ - 1) \cdot M_{2\_} + 1 : ixc\_ \cdot M_{2\_}$$

1.2.2 Average PSNR calculation

This section provides the quality measurements between two video sequences as the average peak signal-to-noise ratio (PSNR) and the true color PSNR (RGB-PSNR). PSNR (1.2) of each video frame is calculated from mean square error (MSE) (1.1) between two video sequences, *y* and $\hat{y}$. Peak value of a video pixel is 255 in (1.2) for 8 bit per pixel component of a video sequence. The average PSNR over *F* frames is given in (1.3).

$$\mathrm{MSE}(p) = \frac{1}{Yrowh \cdot Ycolw} \sum_{i=0}^{Yrowh-1} \sum_{j=0}^{Ycolw-1} \left[y(i, j, p) - \hat{y}(i, j, p) \right]^2 , \tag{1.1}$$

$$\mathrm{PSNR}(p) = 10 \log_{10} \left[\frac{255^2}{\mathrm{MSE}(p)} \right], \tag{1.2}$$

$$\text{average PSNR} = 10 \log_{10} \left[\frac{255^2}{\frac{1}{F} \sum_{p=1}^{F} \mathrm{MSE}(p)} \right], \tag{1.3}$$

where, $y(i, j, p)$ is a pixel at i th row, j th column in the p th frame of a video y. $\hat{y}(i, j, p)$ is a pixel at i th row, j th column in the p th frame of a video $\hat{y}$.

The RGB-PSNR given in (1.5) is computed from MSE_{RGB} in (1.4). MSE_{RGB} is the average MSE of all three RGB video components (R-red, B-green, and B-blue) for each true color video frame.

$$\begin{aligned} \mathrm{MSE}_{\mathrm{RGB}}(p) = \frac{1}{3 \cdot Yrowh \cdot Ycolw} \Bigg\{ & \sum_{i=0}^{Yrowh-1} \sum_{j=0}^{Ycolw-1} \left[R(i, j, p) - \hat{R}(i, j, p) \right]^2 + \\ & + \sum_{i=0}^{Yrowh-1} \sum_{j=0}^{Ycolw-1} \left[G(i, j, p) - \hat{G}(i, j, p) \right]^2 + \sum_{i=0}^{Yrowh-1} \sum_{j=0}^{Ycolw-1} \left[B(i, j, p) - \hat{B}(i, j, p) \right]^2 \Bigg\} , \end{aligned} \tag{1.4}$$

$$\text{RGB-PSNR} = 10 \log_{10} \left[\frac{255^2}{\frac{1}{F} \sum_{p=1}^{F} \mathrm{MSE}_{\mathrm{RGB}}(p)} \right], \tag{1.5}$$

1.2.3 Bit rate calculations

Bit rate in bits per second (bps), of each encoded frame is calculated from the product of encoded bits in one frame and the frame rate in frames per second (fps). Average bit rate (1.6) is the product of the total encoded bits and the frame rate divided by the number of frames, *F*.

$$BitRate(\text{in bps}) = \frac{TotalBit(\text{in bits})}{F(\text{number of frames})} \times FrameRate(\text{in fps}), \tag{1.6}$$

In addition, bit rate in bits per pixel (bpp) (1.7), can be calculated from the average encoded file size in bits in one frame divided by the number of pixels in one video frame (*FrameSize*).

$$BitRate\left(\text{in bpp}\right) = \frac{FileSize\left(\text{in Bytes}\right)\times 8}{F\left(\text{number of frames}\right)\times FrameSize\left(\text{in pixels}\right)}, \qquad (1.7)$$

1.2.4 Definition of structural similarity (SSIM) index

The structural similarity (SSIM) index [13] is a new objective method to measure image quality between a distorted image and a reference image. SSIM measures the degradation of structural information based on the assumption that the human visual system (HVS) characteristics are highly adapted for extracting structural information from an image scene. The authors [13] claim that SSIM has better consistency with perceived image quality than pixel error model (absolute differences and MSE) based on different performance evaluations.

The authors [13] provide the following observations on the nature of natural images and human visual characteristics.

1. Natural images are spatially non-stationary in general.

2. Image distortions can be space-variant. The distortions may or may not depend on local image statistics.

3. Only a local area in the image can be perceived by humans at one time instance, regarding to HVS.

4. Localized quality measurement of the image provides a spatially varying quality map (called SSIM map) of the image which provides more quality degradation information of the image.

SSIM (1.8) is defined as the product of three local quantities: the luminance comparison $l(\underline{x},\underline{y})$, the contrast comparison $c(\underline{x},\underline{y})$, and the structure comparison $s(\underline{x},\underline{y})$.

$$\text{SSIM}\left(\underline{x},\underline{y}\right) = \left[l\left(\underline{x},\underline{y}\right)\right]^{\alpha}\cdot\left[c\left(\underline{x},\underline{y}\right)\right]^{\beta}\cdot\left[s\left(\underline{x},\underline{y}\right)\right]^{\gamma}, \qquad (1.8)$$

where, $l\left(\underline{x},\underline{y}\right)=\frac{2\mu_x\mu_y+C_1}{\mu_x^2+\mu_y^2+C_1}$, $c\left(\underline{x},\underline{y}\right)=\frac{2\sigma_x\sigma_y+C_2}{\sigma_x^2+\sigma_y^2+C_2}$, and $s\left(\underline{x},\underline{y}\right)=\frac{2\sigma_{xy}+C_3}{\sigma_x\sigma_y+C_3}$,

$$\mu_x=\frac{1}{N^2}\sum_{i=1}^{N^2}x_i$$

$$\sigma_x^2=\frac{1}{N^2-1}\sum_{i=1}^{N^2}\left(x_i-\mu_x\right)^2$$

$$\sigma_{xy}=\frac{1}{N^2-1}\sum_{i=1}^{N^2}\left(x_i-\mu_x\right)\left(y_i-\mu_y\right)$$

$\underline{x}$ and $\underline{y}$ are two local window sizes ($N\times N$) of the same spatial location from two images. x_i is the i th pixel of $\underline{x}$, where $i=1,\cdots,N^2$. μ_x is the mean of $\underline{x}$. σ_x^2 is the variance of $\underline{x}$. σ_{xy} is the covariance of $\underline{x}$ and $\underline{y}$. α, β, and γ are parameters to adjust the three quantities. These three parameters are set to 1 for simplicity [13]. C_1, C_2, and C_3 are constants to prevent an unstable measurement where the numerator of each term is close to zero. Regarding [13], these constants are set as $C_1=\left(K_1L\right)^2$, $C_2=\left(K_2L\right)^2$, $C_3=C_2/2$, $K_1=0.01$, and $K_2=0.03$. $L=255$ is the dynamic range of pixel values (for 8 bits per pixel component).

Some properties of the SSIM index are, [13][39]

1. Symmetry: $\mathrm{SSIM}\left(\underline{x},\underline{y}\right)=\mathrm{SSIM}\left(\underline{y},\underline{x}\right)$
2. Boundedness: $\mathrm{SSIM}\left(\underline{x},\underline{y}\right)\le 1$
3. Unique maximum: $\mathrm{SSIM}\left(\underline{x},\underline{y}\right)=1$ if and only if $\underline{x}=\underline{y}$ (in discrete representations, $x_i=y_i$ for all $i=1,2,\cdots,N^2$)

The mean SSIM (MSSIM) index [13] for a single overall quality value of the entire image, is defined as the average of the SSIM map,

$$\mathrm{MSSIM}\left(\underline{x}, \underline{y}\right) = \frac{1}{M}\sum_{j=1}^{M}\mathrm{SSIM}\left(\underline{x}_j, \underline{y}_j\right), \tag{1.9}$$

where, $\underline{x}_j$ is the j th (8×8) local window of and image. M is the number of local windows.

SSIM is applied in two areas of this research, i.e., calculation of motion block matching distortion, and measurement of reconstructed (decoded) video quality.

1.2.5 Video quality assessment using SSIM

The research measures the perceived video quality based on the method introduced in [15] with modifications. Local window weighting value [15] is set as 1. Frame weighting [15] is set as constant for simplicity.

In this research, the local window size is (11×11) for all three video components. Each local block moves pixel by pixel and gives an SSIM value (1.8). SSIM of each video component, Y-SSIM, Cb-SSIM, and Cr-SSIM is calculated independently, in (1.10)–(1.12).

$$\text{Y-SSIM} = \frac{1}{F \cdot M}\sum_{p=1}^{F}\sum_{j=1}^{M}\mathrm{SSIM}_{pj}^{\mathrm{Y}}, \tag{1.10}$$

$$\text{Cb-SSIM} = \frac{1}{F \cdot M}\sum_{p=1}^{F}\sum_{j=1}^{M}\mathrm{SSIM}_{pj}^{\mathrm{Cb}}, \tag{1.11}$$

$$\text{Cr-SSIM} = \frac{1}{F \cdot M}\sum_{p=1}^{F}\sum_{j=1}^{M}\mathrm{SSIM}_{pj}^{\mathrm{Cr}}, \tag{1.12}$$

where, $\mathrm{SSIM}_{pj}^{\mathrm{Y}}$, $\mathrm{SSIM}_{pj}^{\mathrm{Cb}}$, and $\mathrm{SSIM}_{pj}^{\mathrm{Cr}}$ are the SSIM (index) values at the j th local window in the p th frame of the corresponding Y, Cb, and Cr components. Each SSIM index is computed from (1.8). F is the number of video frames. M is the number of local windows.

RGB-SSIM (SSIM of a reconstructed true color video), is calculated as follows. Reconstructed video sequence in YCbCr 4:2:0 format is converted to YCbCr 4:4:4. Cb and Cr components are upsampled by 2 and then are convolved with 2-D zero-order hold interpolation filter. RGB components are obtained from YCbCr-to-RGB transform matrix on each YCbCr 4:4:4 i th pixel, in (1.13) [17]. The transform matrix is given by [17].

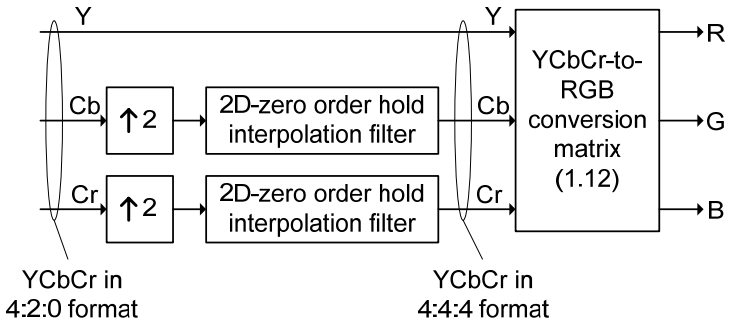


Figure 1.6 Color format conversion for YCbCr 4:2:0 to RGB 4:4:4

$$\begin{bmatrix} r_i \\ g_i \\ b_i \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0 \end{bmatrix} \begin{bmatrix} y_i \\ cb_i \\ cr_i \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}, \tag{1.13}$$

The block diagram of color format conversion is shown in Fig. 1.6. Sampling formats of YCbCr video component, such as, 4:2:0, 4:4:4, 4:2:2, 4:1:1, and 4:0:0 (gray scale), used in video coding standards are shown in Fig. 1.7.

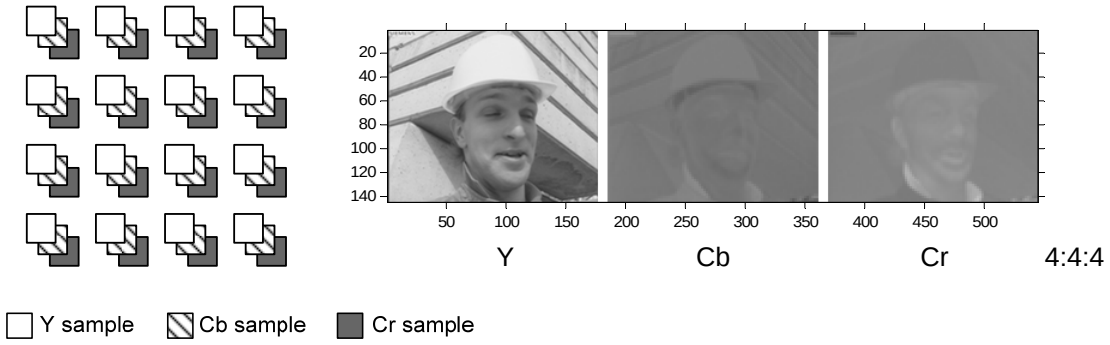
The SSIM of each R, G, and B component is calculated independently by (1.8). This research proposes measuring RGB-SSIM by averaging the three components, in (1.14)

$$\text{RGB-SSIM} = \frac{1}{F \cdot M} \sum_{p=1}^{F} \sum_{j=1}^{M} \left(\frac{1}{3} \text{SSIM}_{pj}^{\text{R}} + \frac{1}{3} \text{SSIM}_{pj}^{\text{G}} + \frac{1}{3} \text{SSIM}_{pj}^{\text{B}} \right), \tag{1.14}$$

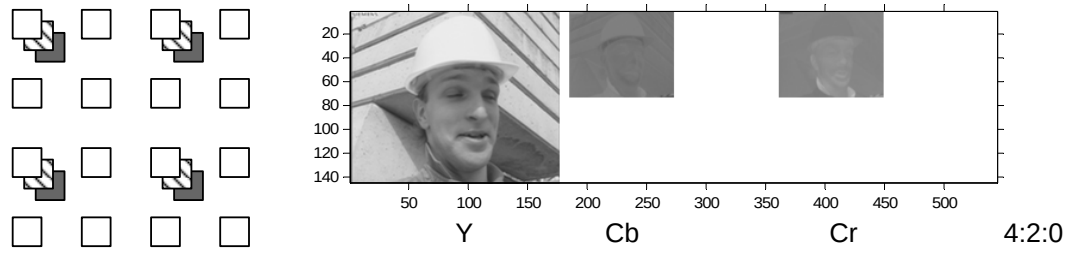
where, $\text{SSIM}_{pj}^{\text{R}}$, $\text{SSIM}_{pj}^{\text{G}}$, and $\text{SSIM}_{pj}^{\text{B}}$ are the SSIM values (1.8) at the j th local window in the p th video frame of the corresponding R, G, and B components. Each SSIM index is computed from (1.8). F is the number of video frames. M is the number of local windows.

1.2.6 Rate-distortion curve comparison using BD-PSNR

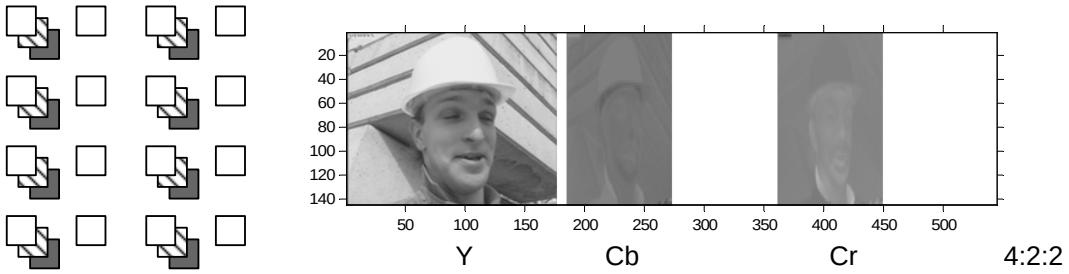
BD-PSNR (Bjontegaard delta peak signal-to-noise ratio) [18] is a single number representing the average PSNR difference (in dB) between two rate-distortion (RD) plots (PSNR versus bit rate). Microsoft Office Excel Add-In for BD-PSNR can be obtained from [19]. To obtain BD-PSNR between two rate-distortion (RD) curves, each RD plot is modeled by fitting the RD curve with four data pairs as,



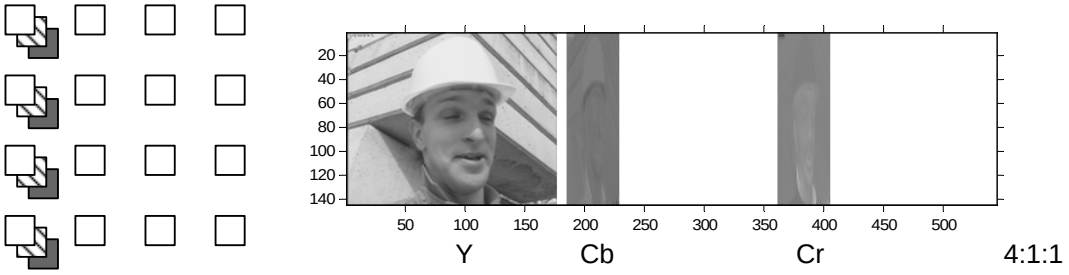
(a)



(b)



(c)



(d)

Figure 1.7 Sampling formats on a block of (4×4) pixels of YCbCr video sequence (left) and (right) the corresponding components of Foreman QCIF (144×176), frame 1. (a) 4:4:4, (b) 4:2:0, (c) 4:2:2, (d) 4:1:1

$$\text{PSNR} = a + b \cdot x + c \cdot x^2 + d \cdot x^3$$

where, x is the logarithmic bit rate given by $\log_e(\text{bit rate})$. $\{a, b, c, d\}$ are parameters used for curve fitting. BD-PSNR between RD plot 0 and RD plot 1 is given in (1.15),

$$\text{BD-PSNR} = \left[\left(a_1\Delta x + b_1\Delta x^2 + c_1\Delta x^3 + d_1\Delta x^4\right) - \left(a_0\Delta x + b_0\Delta x^2 + c_0\Delta x^3 + d_0\Delta x^4\right)\right] / \Delta x \,, \quad (1.15)$$

where, Δx is the difference between the maximum and minimum of logarithmic bit rates. Δx^m is the difference between the maximum bit rate with power m and minimum bit rate with power m. The author [18] also provides the calculation of BD-bit rate, the average bit rate difference in percentage between two RD plots.

1.3 Variable Block Size Transform in Block Based Hybrid Video Coding

Variable block size transform coding is a coding method that utilizes different transform sizes to compensate the effect of using a fixed-size transform. A large transform provides a better energy compaction and a better preservation of detail in a quantized block than a small transform does [20]. Trends, textures and periodic textures are better preserved with the transforms having a larger size. A smaller transform is better in areas with discontinuities because it produces fewer ringing artifacts [21].

In video coding standards, only H.264 and Windows Media Video 9 (WMV-9) [21][22] use a variable block size transform. Audio Video coding Standard of China (AVS-China) uses fixed size integer transforms: (8×8), for AVS Part 2 (AVS-video 1.0, for high-definition digital video broadcasting and high-density storage media) and (4×4) for AVS Part 7 (AVS-M, for low complexity and low picture resolution mobile applications) [23][24]. Other standards, such as, MPEG-4 Visual, H.263, MPEG-2 (H.262), MPEG-1, and H.261 use fixed-size (8×8) discrete cosine transform (DCT) [3][4].

In H.264, three profiles including baseline, main, and extended profiles use only a (4×4) integer transform. In H.264 fidelity range extensions (FRExt) [25], either a (8×8) or a (4×4) integer transform is used. Variable block size transform coding was proposed in [20]. It is called

the adaptive block size transform (ABT). ABT is not applied to H.264. Here, (8×8) and (4×4) integer transforms are used to transform the residual block sizes (8×8), (8×4), (4×8), and (4×4). The ABT 2-D forward transform is defined as [20],

$$Y = T_M \cdot X \cdot T_N^T \,, \tag{1.16}$$

where, X is the residual block size of (M×N). Y is the transform coefficient matrix. T_M and T_N are (M×M) and (N×N) transform matrices. M can be either 4 or 8, and N can be either 4 or 8. T_N^T is the transpose of T_N.

WMV-9 uses (8×8) and (4×4) integer discrete cosine transforms, where forward and inverse transforms form an orthogonal pair. The forward transform is defined as [22],

$$Y = \left(T_M \cdot X \cdot T_N^T\right) \otimes W_{NM}$$

where, ⊗ denotes the scalar multiplication of corresponding elements of the two matrices (not matrix multiplication). X is a pixel block of (M×N) for intra coding, or a residual block of (M×N) for inter coding. Y is the transform coefficient matrix. W_{NM} are the normalization matrices defined in [22]. M is 4 or 8, and N is 4 or 8.

For inter predicted residual blocks, WMV-9 uses (8×8), (8×4), (4×8), and (4×4) block transforms. The transform size and shape are best suited for the underlying residual block. Intra frames and intra blocks (macroblocks) in predicted frames use fixed 8×8 transforms [21].

In the literature [5], variable block size transform coding is based on a quadtree structure of DCT blocks. The choice of block size is based on the mean-based decision rule, (the mean difference of the four adjacent blocks and the prescribed threshold). In [6], the coding method is also a quadtree based variable block size DCT. Bit count values of a transform block and the corresponding four sub-blocks are calculated. Transform coefficient block size is, then, determined by the block or sub-block that has smaller bit count values.

1.4 Structural Similarity Index in Video Codec Design

Two papers [26][27] applied SSIM in JM/H.264 reference software. In [26], the author claims that motion estimation method is improved using distortion based on structural similarity (MEBSS, motion estimation method based on structural similarity). Both bit rate and coding time are reduced. The simulation is done only at quantization parameter (*QP*) = 10, (low *QP* value, high quality, high bit rate). Distortion cost for motion block matching metric is changed from absolute differences to (1 – SSIM) with full search motion estimation. SSIM is computed with local block size (4×4) non-overlapped windows.

In [27], the authors propose another technique to reduce the coding time in motion estimation. It is called fast MEBSS (FMEBSS). SSIM is used in an early termination algorithm (with fixed threshold value) and distortion cost is applied where absolute differences are changed to $K(1-\mathrm{SSIM})$. *K* is the user defined multiplier obtained by intensive experiments. The simulation is done at *QP* = 10, 20, and 30. The codec achieves bit rate and coding time reductions compared to ME based on absolute differences.

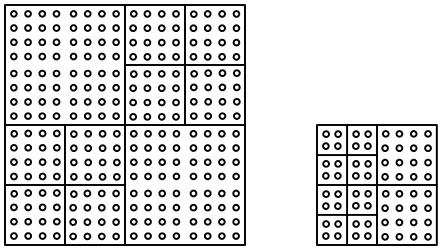
1.5 Organization of the Dissertation

The dissertation is organized as follows. Chapter 2 provides the details of variable block size transform coding. Chapter 3 discusses interframe coding, motion estimation with SAD and SSIM in the motion block matching. Simulation results are discussed in Chapter 4. Chapter 5 provides summary and future work.

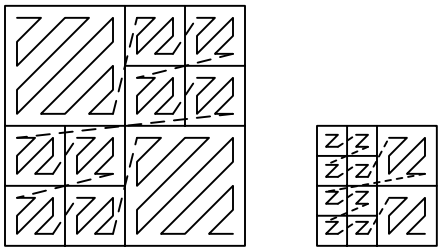
CHAPTER 2

INTRAFRAME CODING WITH VARIABLE SIZE INTEGER TRANSFORMS

In the intraframe encoding process (Fig. 1.2), a video frame is partitioned into (16×16)-pixel luminance macroblocks and (8×8)-pixel chrominance macroblocks. Each macroblock is spatially predicted in pixel domain (from previously reconstructed macroblocks) and forms the residual macroblock. Each residual macroblock is transformed and quantized with a variable block size transform encoder. The results are a composite coefficient block (a combination of different sub-blocks) and block-size selection bits. The composite block is performed sub-block zig-zag scan (Fig. 2.1), run-length encoding, and Huffman encoding. Finally, the bit stream is formed.



(a)



(b)

Figure 2.1 An example of sub-block zig-zag scan with respect to (16×16) luma and (8×8) chroma composite blocks. (a) a composite block, (b) the corresponding sub-block zig-zag scan

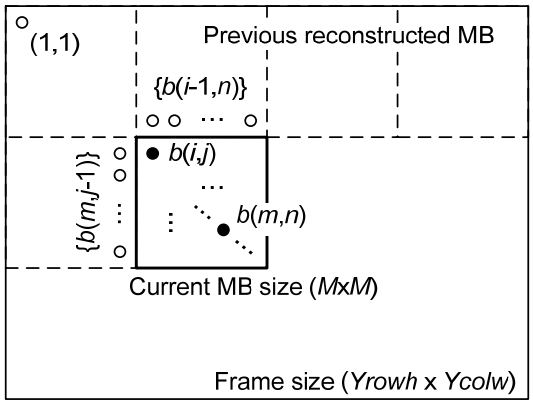


Figure 2.2 Pixel locations for spatial mode predictions

2.1 H.264-Like Spatial Prediction

The proposed codec applies (16×16)-spatial prediction (similar to [1] [4]) on each luminance macroblock, and only the average mode (dc mode) on each chrominance macroblock. The prediction of a current block is based on the pixels of the previous reconstructed blocks (Fig. 2.2).

In Figure 2.2, let (1,1) be the first pixel index (top-left corner) of a video frame with size $Yrowh \times Ycolw$. A current macroblock ($M \times M$) has its first pixel (top-left corner) located at (i, j). $b(m,n)$ is a pixel of a current macroblock located at the m th-row, n th-column. $M = 16$ is used for a luma macroblock, and $M = 8$ is used for a chroma macroblock. $pred_S(m,n)$ is a pixel of a predicted macroblock. s is a spatial mode number, $s \in \{0,1,2,3\}$. ('0' is for vertical mode. '1' is for horizontal mode. '2' is for dc mode. '3' is for plane mode.) The spatial modes are defined in (2.1)–(2.4),

$$pred_0(m,n) = b(i-1,n), \quad \text{when } \{b(i-1,n)\} \text{ is available, } n = j,\cdots,j+M-1,\ m = i,\cdots,i+M-1, \tag{2.1}$$

$$pred_1(m,n) = b(m,j-1), \quad \text{when } \{b(m,j-1)\} \text{ is available, } m = i,\cdots,i+M-1,\ n = j,\cdots,j+M-1, \tag{2.2}$$

$$pred_2(m,n)=\begin{cases}\text{round}\left(\dfrac{1}{2M}\left[\displaystyle\sum_{n=j}^{j+M-1}b(i-1,n)+\sum_{m=i}^{i+M-1}b(m,j-1)\right]\right),\\ \qquad\text{when } \{b(i-1,n)\} \text{ and } \{b(m,j-1)\} \text{ are available}\\ \text{round}\left(\dfrac{1}{M}\displaystyle\sum_{n=j}^{j+M-1}b(i-1,n)\right), \text{when } \{b(i-1,n)\} \text{ are available}\\ \text{round}\left(\dfrac{1}{M}\displaystyle\sum_{m=i}^{i+M-1}b(m,j-1)\right), \text{when } \{b(m,j-1)\} \text{ are available}\\ 128, \quad \text{otherwise}\end{cases}, \tag{2.3}$$

where, $m=i,\cdots,i+M-1$, $n=j,\cdots,j+M-1$

$$pred_3(m,n)=\begin{cases}0, & g(m,n)<0\\ 255, & g(m,n)>255\\ g(m,n), & \text{otherwise}\end{cases}, \tag{2.4}$$

where,

$$g(m,n)=\frac{1}{32}\left[d+e_H\{\text{mod}(n-1,M)-7\}+f_V\{\text{mod}(m-1,M)-7\}\right],$$

$$d=M\left[b(i+M-1,j-1)+b(i-1,j+M-1)\right],$$

$$e_H=\text{round}\left(\frac{5}{64}\left\{\sum_{a=1}^{8}a\left[b(i-1,7+j+a)-b(i-1,7+j-a)\right]\right\}\right),$$

$$f_V=\text{round}\left(\frac{5}{64}\left\{\sum_{a=1}^{8}a\left[b(7+i+a,j-1)-b(7+i-a,j-1)\right]\right\}\right)$$

and, round(x) is the rounding function which rounds its value x to the nearest integer.

An example of a spatial predicted frame is shown in Fig. 2.3.

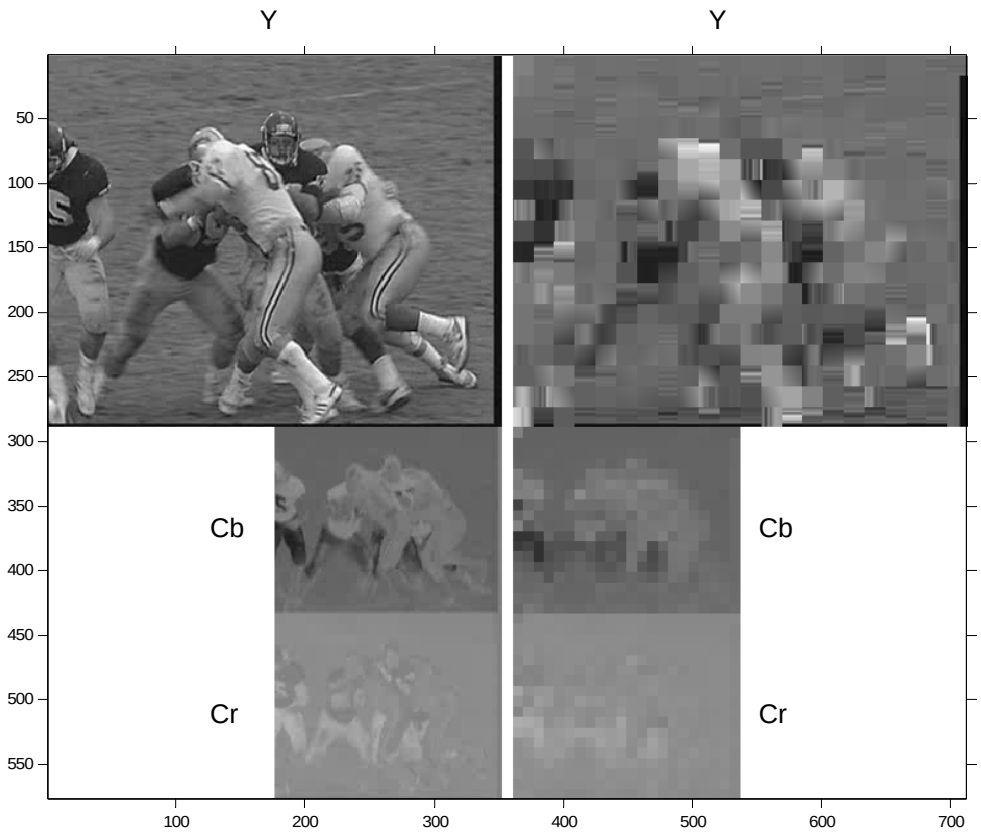


Figure 2.3 Spatial prediction on the reconstructed frame 50 of Football CIF, *QP* = 30. Left column is the reconstructed frame Y, Cb, and Cr. Right column is the spatial predicted frame Y, Cb, and Cr.

2.2 Variable Block Size Transform

The detailed block diagrams of the variable size transform encoder and decoder for both luma and chroma macroblocks are given in Figs. 2.4–2.7. The details on integer transform operations, the quantization scheme, block size selection procedure are discussed in this section.

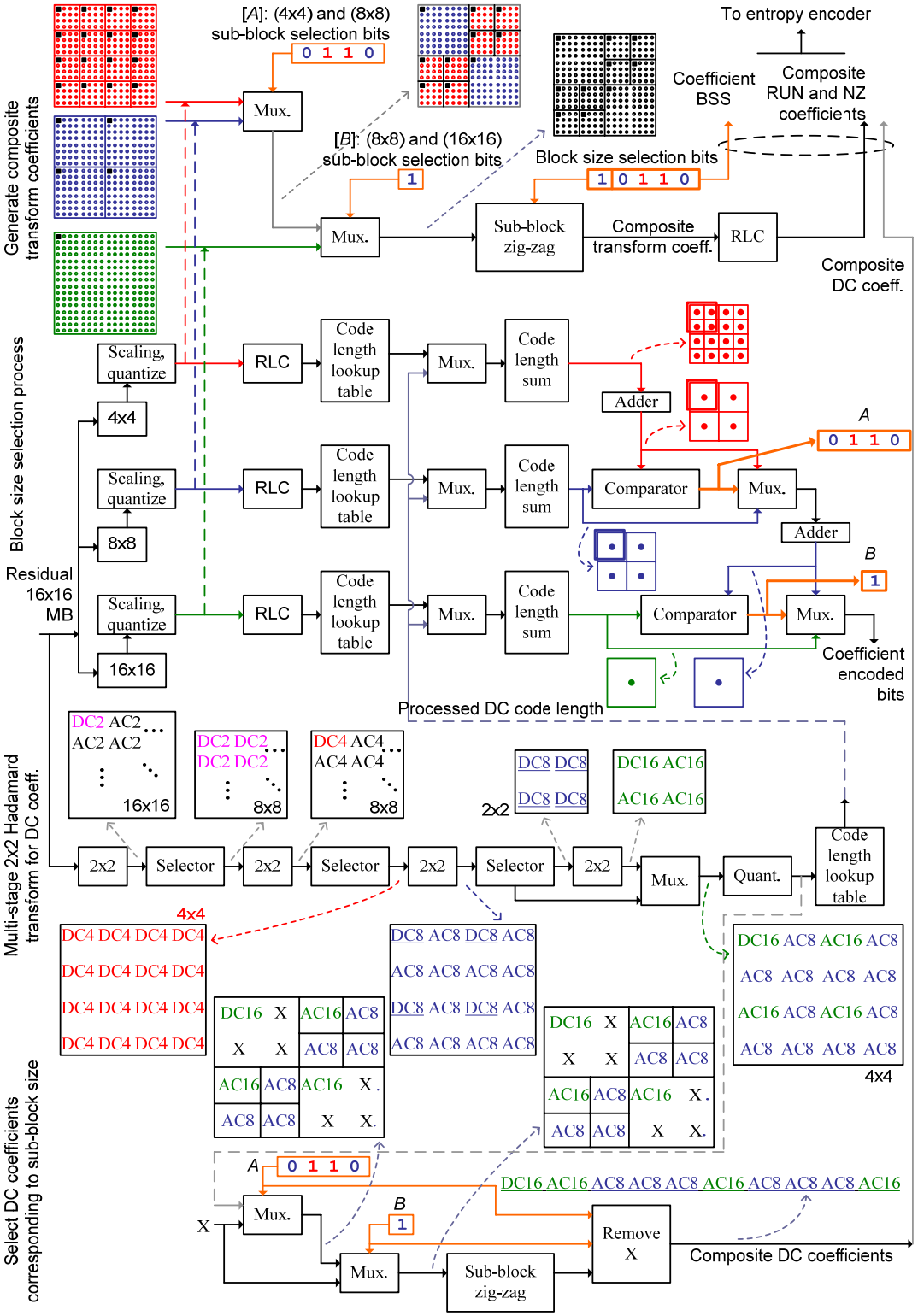


Figure 2.4 Variable size transform encoder block diagram [6] with modifications for a luma block

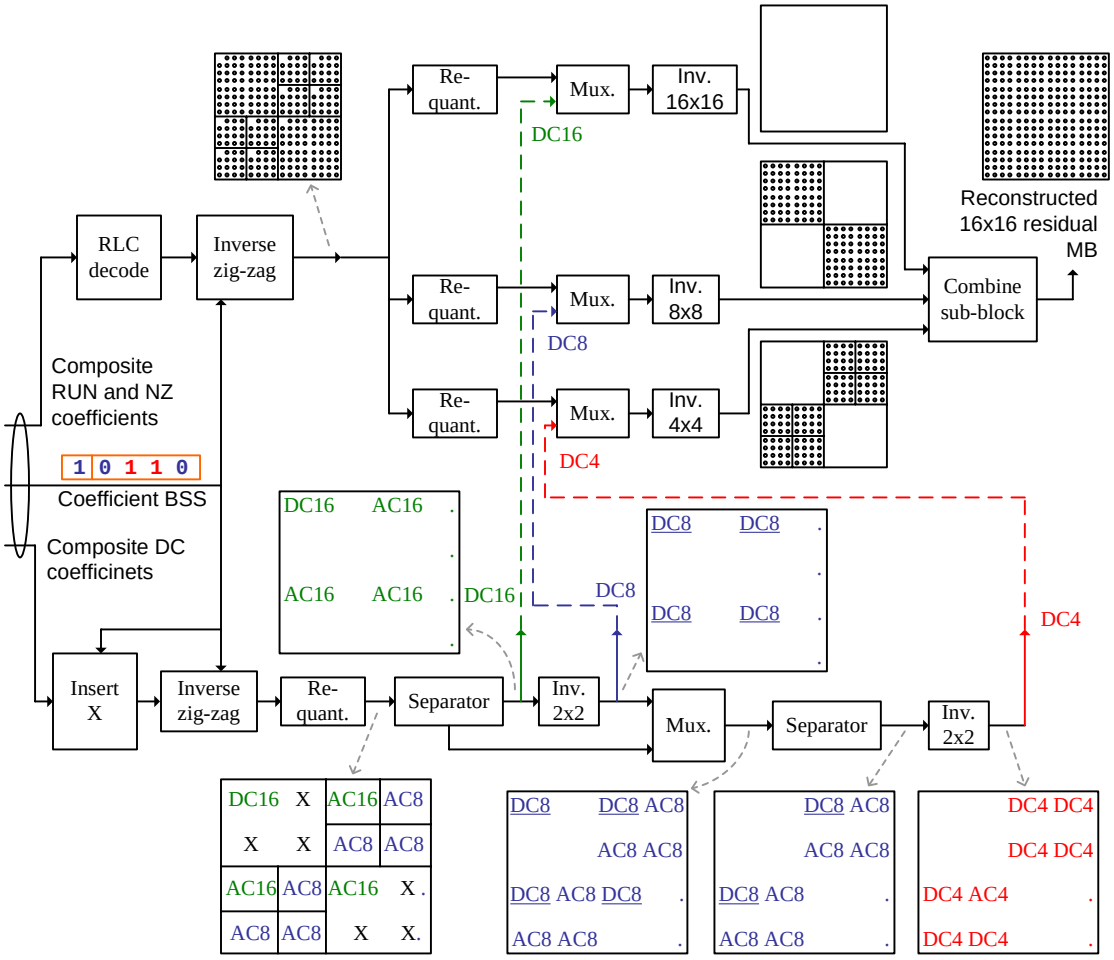


Figure 2.5 Variable size transform decoder block diagram [6] with modifications for a luma block

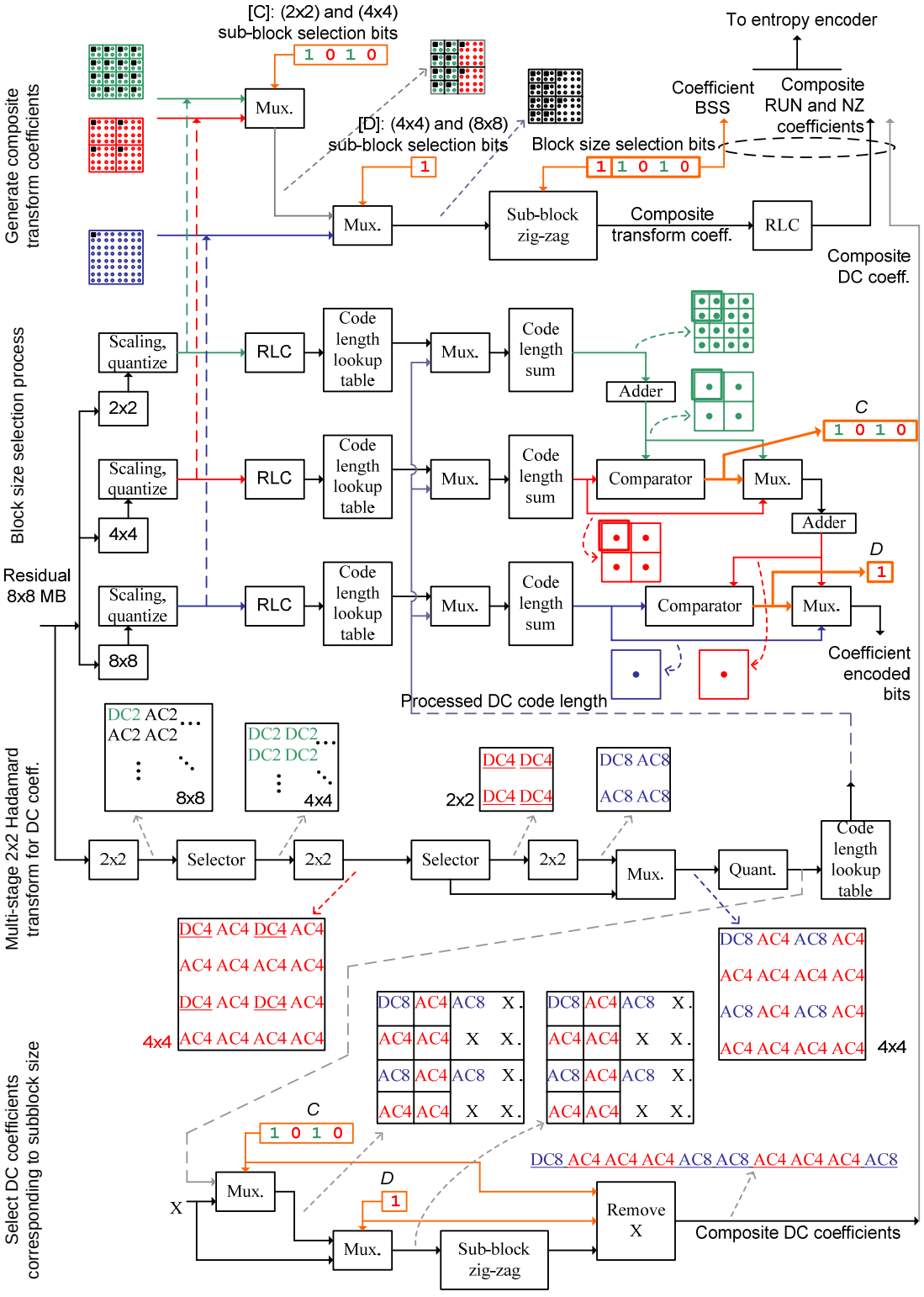


Figure 2.6 Transform encoder block diagram [6] with modifications for a chroma block

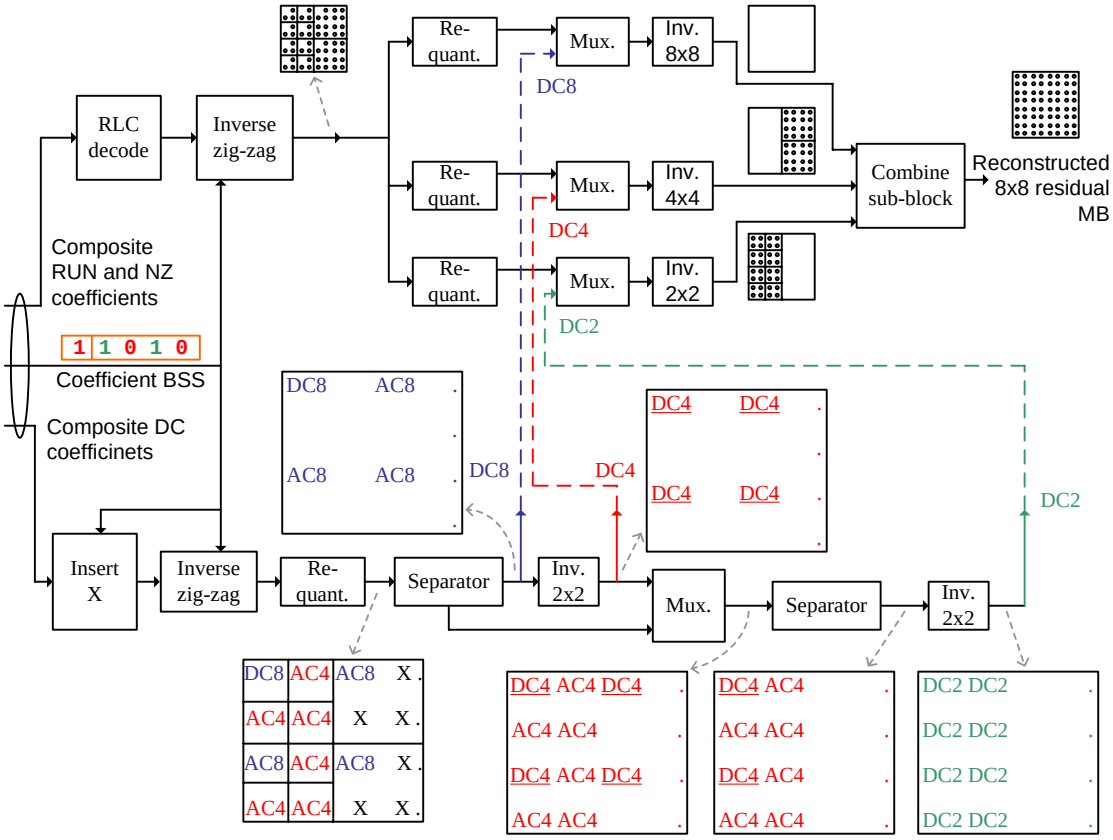


Figure 2.7 Variable size transform decoder block diagram [6] with modifications for a chroma block

2.2.1 Integer transforms and quantization

Four integer transforms are used in the transform encoder (Figs. 2.4, 2.7), (2×2)-Hadamard transform, (4×4) and (8×8) H.264 integer transforms [28][4][29]. For (16×16), either the order-16 integer cosine transform (ICT) [30] or the simplified order-16 ICT [31] can be used. These two integer transforms were designed for general uses in signal and image processing applications. These transforms are applied in the proposed video codec. Performances of these integer transforms are evaluated a transform coding gain, G_{TC} , [32], are,

Table 2.1 Transform Coding Gains of Normalized DCT-II and the Integer Approximations, on Markov-I Process with Adjacent Correlation Coefficient 0.9

Size	Normalized DCT-II	Integer approximation
2×2	3.606 dB	3.606 dB (Hadamard transform)
4×4	5.387 dB	5.375 dB (H.264 integer transform) [28]
8×8	6.276 dB	6.237 dB (H.264 integer transform)
16×16	6.726 dB	6.684 dB (Order-16 integer cosine transform) 6.540 dB (Simplified order-16 ICT)

$$G_{TC} = \frac{\text{Arithmetic mean}}{\text{Geometric mean}} = \frac{\frac{1}{N}\sum_{i=0}^{N-1}\tilde{\sigma}_{ii}^{2}}{\left(\prod_{i=0}^{N-1}\tilde{\sigma}_{ii}^{2}\right)^{\frac{1}{N}}}, \tag{2.5}$$

where, N is the transform size, $\tilde{\sigma}_{ii}^{2}$ is the variance of the i th transform coefficient of the covariance matrix.

Transform coding gain comparison of these integer transforms with the normalized discrete cosine transform of type two (DCT-II) is given in Table 2.1. The covariance matrix is generated by a first order Markov process (Markov-1 process). Adjacent correlation coefficient is set to 0.9.

From Table 2.1, the performances of these transforms are close to the DCT-II. In the case of a (16×16) transform size, the order-16 ICT has transform coding gain higher than the simplified order-16 ICT. However, [31] claims that the simplified order-16 ICT can be implemented without multiplies (only adds and shifts).

In the coding procedure, (4×4), (8×8), and (16×16) transforms are applied to the luma residue macroblock. For a chroma residue macroblock, (2×2), (4×4), and (8×8) transforms are used. The forward transforms Y_N of each sub-block X_N with size (N×N), where N = 2, 4, 8, and 16, are as follows,

$$Y_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} X_2 \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}, \tag{2.6}$$

$$Y_4 = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X_4 \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} \frac{1}{4} & \frac{1}{\sqrt{40}} & \frac{1}{4} & \frac{1}{\sqrt{40}} \\ \frac{1}{\sqrt{40}} & \frac{1}{10} & \frac{1}{\sqrt{40}} & \frac{1}{10} \\ \frac{1}{4} & \frac{1}{\sqrt{40}} & \frac{1}{4} & \frac{1}{\sqrt{40}} \\ \frac{1}{\sqrt{40}} & \frac{1}{10} & \frac{1}{\sqrt{40}} & \frac{1}{10} \end{bmatrix}, \tag{2.7}$$

$$Y_8 = \Lambda_8 T_8 X_8 T_8^T \Lambda_8 , \tag{2.8}$$

$$Y_{16} = \Lambda_{16} T_{16} X_{16} T_{16}^T \Lambda_{16} , \tag{2.9}$$

where, $\otimes$ denotes the scalar multiplication of corresponding elements (not matrix multiplication).

T_8 is (8x8) H.264 integer transform [29].

$$T_8 = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix},$$

$$\Lambda_8 = \text{diag}\left\{ \frac{1}{16\sqrt{2}}, \frac{1}{17\sqrt{2}}, \frac{1}{8\sqrt{5}}, \frac{1}{17\sqrt{2}}, \frac{1}{16\sqrt{2}}, \frac{1}{17\sqrt{2}}, \frac{1}{8\sqrt{5}}, \frac{1}{17\sqrt{2}} \right\}$$

For the order-16 ICT [30],

$$T_{16} = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
42 & 38 & 37 & 32 & 22 & 19 & 10 & 4 & -4 & -10 & -19 & -22 & -32 & -37 & -38 & -42 \\
55 & 48 & 32 & 11 & -11 & -32 & -48 & -55 & -55 & -48 & -32 & -11 & 11 & 32 & 48 & 55 \\
38 & 22 & 4 & -19 & -37 & -42 & -32 & -10 & 10 & 32 & 42 & 37 & 19 & -4 & -22 & -38 \\
3 & 1 & -1 & -3 & -3 & -1 & 1 & 3 & 3 & 1 & -1 & -3 & -3 & -1 & 1 & 3 \\
37 & 4 & -32 & -38 & -10 & 22 & 42 & 19 & -19 & -42 & -22 & 10 & 38 & 32 & -4 & -37 \\
48 & -11 & -55 & -32 & 32 & 55 & 11 & -48 & -48 & 11 & 55 & 32 & -32 & -55 & -11 & 48 \\
32 & -19 & -38 & 4 & 42 & 10 & -37 & -22 & 22 & 37 & -10 & -42 & -4 & 38 & 19 & -32 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
22 & -37 & -10 & 42 & -4 & -38 & 19 & 32 & -32 & -19 & 38 & 4 & -42 & 10 & 37 & -22 \\
32 & -55 & 11 & 48 & -48 & -11 & 55 & -32 & -32 & 55 & -11 & -48 & 48 & 11 & -55 & 32 \\
19 & -42 & 22 & 10 & -38 & 32 & 4 & -37 & 37 & -4 & -32 & 38 & -10 & -22 & 42 & -19 \\
1 & -3 & 3 & -1 & -1 & 3 & -3 & 1 & 1 & -3 & 3 & -1 & -1 & 3 & -3 & 1 \\
10 & -32 & 42 & -37 & 19 & 4 & -22 & 38 & -38 & 22 & -4 & -19 & 37 & -42 & 32 & -10 \\
11 & -32 & 48 & -55 & 55 & -48 & 32 & -11 & -11 & 32 & -48 & 55 & -55 & 48 & -32 & 11 \\
4 & -10 & 19 & -22 & 32 & -37 & 38 & -42 & 42 & -38 & 37 & -32 & 22 & -19 & 10 & -4
\end{bmatrix},$$

and,

$$\Lambda_{16} = \operatorname{diag}\left\{\frac{1}{4}, \frac{1}{2\sqrt{3281}}, \frac{1}{2\sqrt{6474}}, \frac{1}{2\sqrt{3281}}, \frac{1}{4\sqrt{5}}, \frac{1}{2\sqrt{3281}}, \frac{1}{2\sqrt{6474}}, \frac{1}{2\sqrt{3281}}, \right.$$
$$\left. \frac{1}{4}, \frac{1}{2\sqrt{3281}}, \frac{1}{2\sqrt{6474}}, \frac{1}{2\sqrt{3281}}, \frac{1}{4\sqrt{5}}, \frac{1}{2\sqrt{3281}}, \frac{1}{2\sqrt{6474}}, \frac{1}{2\sqrt{3281}}\right\}$$

For the simplified order-16 ICT (ICT_{16}) to be used in (2.9), the flow diagram (signal flow graph) given in [27] can be converted to sparse matrix factors (SMF) (2.11). The permutation matrices, $P1$ and $P2$ are introduced to keep both data and transformed sequences in natural order. Therefore, T_{16} is given as,

$$T_{16} = [P2][ICT_{16}][P1], \tag{2.10}$$

where,

$$ICT_{16} = \begin{bmatrix} T_8^{AVS} & \underline{0}_{8\times 8} \\ \underline{0}_{8\times 8} & [M_2 \times M_3 \times M_4] \end{bmatrix} \begin{bmatrix} I_8 & I_8 \\ I_8 & -I_8 \end{bmatrix}, \tag{2.11}$$

$\underline{0}_{8\times8}$ is a (8×8) null matrix, I_8 is an identity matrix of size (8×8). T_8^{AVS} is the (8×8) integer DCT adopted by AVS [23].

$$T_8^{AVS} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 10 & 9 & 6 & 2 & -2 & -6 & -9 & -10 \\ 10 & 4 & -4 & -10 & -10 & -4 & 4 & 10 \\ 9 & -2 & -10 & -6 & 6 & 10 & 2 & -9 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -10 & 2 & 9 & -9 & -2 & 10 & -6 \\ 4 & -10 & 10 & -4 & -4 & 10 & -10 & 4 \\ 2 & -6 & 9 & -10 & 10 & -9 & 6 & -2 \end{bmatrix},$$

$$M_2 \times M_3 \times M_4 = \begin{bmatrix} -2 & 0 & 1 & -1 & -1 & 3 & -1 & 0 \\ 3 & -1 & 1 & 1 & 0 & 2 & 0 & 1 \\ -1 & -3 & 1 & 0 & 1 & 0 & 2 & -1 \\ 0 & 1 & 0 & 1 & 3 & 1 & -1 & -2 \\ 1 & -1 & -3 & -2 & 0 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & -2 & 0 & 1 & -3 \\ 0 & -2 & 0 & 1 & -1 & -1 & -3 & -1 \\ -1 & 0 & -2 & 3 & -1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -2 & 0 & -1 & 0 & 2 & -1 & 1 \\ 0 & -1 & 0 & 2 & 1 & -1 & 0 & 2 \\ 0 & -2 & 1 & 1 & 0 & 0 & 1 & -2 \\ -2 & 0 & -1 & 0 & 2 & 0 & -1 & -1 \\ -1 & 0 & 2 & 0 & -1 & -1 & -2 & 0 \\ -2 & 0 & 1 & -1 & 0 & 0 & 2 & 1 \\ -1 & 1 & 0 & 2 & -1 & 2 & 0 & 0 \\ -1 & -1 & -2 & 0 & -2 & -1 & 0 & 0 \end{bmatrix},$$

$$P2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$P1 = \begin{bmatrix} I_8 & \underline{0}_{8\times 8} \\ \underline{0}_{8\times 8} & \begin{bmatrix} 0 & \cdots & 0 & 1 \\ \vdots & & \iddots & 0 \\ 0 & 1 & & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}_{8\times 8} \end{bmatrix},$$

$$\Lambda_{16} = \mathrm{diag}\left\{\frac{1}{32}, \frac{1}{\sqrt{1122}}, \frac{1}{32}, \frac{1}{\sqrt{1122}}, \frac{1}{4\sqrt{58}}, \frac{1}{\sqrt{1122}}, \frac{1}{2\sqrt{221}}, \frac{1}{\sqrt{1122}}, \right.$$
$$\left. \frac{1}{2\sqrt{221}}, \frac{1}{\sqrt{1122}}, \frac{1}{4\sqrt{58}}, \frac{1}{\sqrt{1122}}, \frac{1}{2\sqrt{221}}, \frac{1}{\sqrt{1122}}, \frac{1}{2\sqrt{221}}, \frac{1}{\sqrt{1122}}\right\}$$

The forward quantized block Z_N, is given by,

$$Z_N = \mathrm{round}\left(\frac{Y_N}{Stepsize}\right), \tag{2.12}$$

The rescaling or inverse quantized block $\hat{Y}_N$ is given by,

$$\hat{Y}_N = Z_N \cdot Stepsize, \tag{2.13}$$

The inverse transforms of each rescaling block are follows,

$$\hat{X}_2 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \hat{Y}_2 \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{2.14}$$

$$\hat{X}_4 = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \mathrm{diag}\left\{\frac{1}{4}, \frac{1}{5}, \frac{1}{4}, \frac{1}{5}\right\} \left(\hat{Y}_4 \otimes \begin{bmatrix} 4 & \sqrt{40} & 4 & \sqrt{40} \\ \sqrt{40} & 10 & \sqrt{40} & 10 \\ 4 & \sqrt{40} & 4 & \sqrt{40} \\ \sqrt{40} & 10 & \sqrt{40} & 10 \end{bmatrix} \right) \mathrm{diag}\left\{\frac{1}{4}, \frac{1}{5}, \frac{1}{4}, \frac{1}{5}\right\} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

(2.15)

$$\hat{X}_8 = T_8^T \Lambda_8 \hat{Y}_8 \Lambda_8 T_8, \tag{2.16}$$

$$\hat{X}_{16} = T_{16}^T \Lambda_{16} \hat{Y}_{16} \Lambda_{16} T_{16}, \tag{2.17}$$

Table 2.2 Quantization Parameters and Quantization Step Sizes

QP	30	31	32	33	34	35	36	37	38	39	40	41	42
Stepsize	20	22	26	28	32	36	40	44	52	56	64	72	80

The uniform quantization is used in the codec which is similar to the H.264 quantization [3][4]. In Table 2.2, the *Stepsize* value is doubled for every sixth quantization parameter, (*QP*).

2.2.2 Block size selection

In Figures 2.4 and 2.6, after the transformed residual blocks are quantized and run-length encoded, the bock size selection process is performed. Based on [6], the decision on a combination of sub-block sizes is based on choosing the minimum encoded bit length of four adjacent sub-blocks $X_N(k,i)$. In Figure 2.8, $X_N(k,i)$ is the i th ($N\times N$) sub-block co-located at the larger k th ($2N\times 2N$) sub-block, $X_{2N}(k)$. $i = 1,\cdots,4$ and $N = 4,8$ for luma block ($N = 2,4$ for chroma block). Each decision produces a decision bit '0' if the number of encoded bits from the large-transformed block is less than the number of encoded bits from the small-transformed blocks. Bit '1' is selected if the number of encoded bits from the small-transformed blocks is less. Block size decision is given in (2.18). Figure 2.9 shows an example of block size selection of a given composite luma macroblock.

$$\sum_{i=1}^{4}\text{CodeLength}\left[X_N(k,i)\right]\begin{matrix}\geq \text{CodeLength}\left[X_{2N}(k)\right], \text{ set '0' and select larger size}\\ < \text{CodeLength}\left[X_{2N}(k)\right], \text{ set '1' and select smaller size}\end{matrix}, \qquad (2.18)$$

where, CodeLength is the number of encoded bits by looking-up the entropy codebook table of the corresponding transformed block.

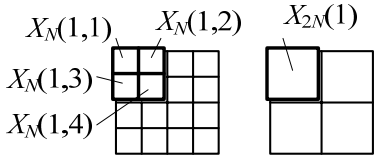


Figure 2.8 An example of four sub-blocks X_N and a co-located larger sub-block X_{2N} in the block selection process

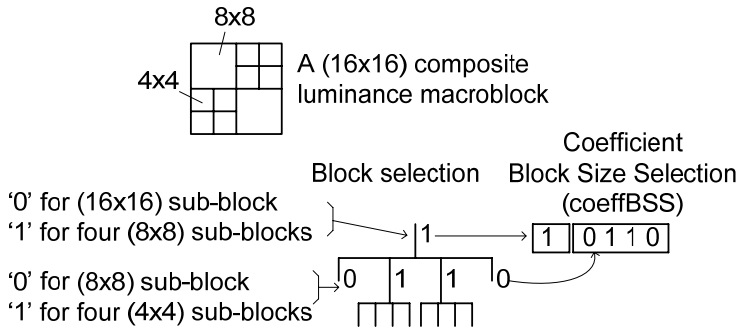


Figure 2.9 An example of block size selection on a composite luminance macroblock

The combined decision bits of a macroblock form a **coeffBSS** (coefficient block size selection), header of a macroblock. This is to identify a macroblock structure. The proposed codec applies three transform sizes to a selection process with two selection stages (Fig. 2.9). The header length of each encoded macroblock is either 1 or 5 bits. In [6], four transform sizes with three selection levels are used and the longer header length is in $\{1,5,9,13,17,21\}$ bits.

2.2.3 Processing dc coefficient

The dc coefficients are processed separately. Multi-stages of a (2×2) Hadamard transform are applied to a residual macroblock [6] (Figs 2.4 and 2.6). The result is the composite dc block from which the dc coefficients of different transform block sizes can be derived. With the **coeffBSS** bits from the block size selection, the selected composite dc coefficients are obtained and encoded. The dc coefficients from this method have average encoded bit-length less than the ones without additional multi-stages Hadamard transforms [6].

2.3 Bit Stream Format

The outputs from spatial prediction and transform coding are mapped into bits by Huffman variable-length code. These coefficients are **Spatial_mode** (spatial prediction mode), **coeffBSS** (block size selection bits), **dc** (composite dc coefficients), and the coefficients from run-length code (RLC), i.e., **NZ** (non-zero coefficients), and **RUN** (the number of consecutive zero coefficients). The proposed bit stream format is given in Table 2.3. End of non-zero symbol

Table 2.3 The Bit Stream Format of the Proposed Codec for I-Frame Encoding

Macroblock type	Bit stream format
Luma I-MB	**Spatial_mode,** **coeffBSS,** **dc_1 … dc_n,** **NZ_1 RUN_1 … NZ_i RUN_i EndOfNZ,** **…** **NZ_1 RUN_1 … NZ_j RUN_j EndOfNZ**
Chroma I-MB	**coeffBSS,** **dc_1 … dc_m,** **NZ_1 RUN_1 … NZ_k RUN_k EndOfNZ,** **…** **NZ_1 RUN_1 … NZ_p RUN_p EndOfNZ**

EndOfNZ (bit '0') in the bit stream structure is to indicate the end of each sub-block.

2.4 Codebook Training of I-Frame

Five test sequences (Bus, City (ABC), Crew (NASA), Foreman, and Soccer) [33] (Fig. 2.10) are used in the codebook training. Spatial modes, dc coefficients and coefficients from run-length code (RLC) are collected. The coefficients from RLC are collected based on each transformed sizes, (16×16), (8×8), (4×4), and (2×2). Histograms for all coefficient types are created. Huffman codebooks are, then, obtained from Huffman trees of the histograms. The Huffman tree method can be found in [34]. A list of codebook names of the proposed codec is given in Table 2.4. Figures 2.11–2.13 show Huffman codebook of each coefficient including the histogram, codebook range, size of the codebook, average code length and its entropy in bits per symbol. In each Huffman table, the first column is coefficient symbols, and the second column is the corresponding binary codewords.

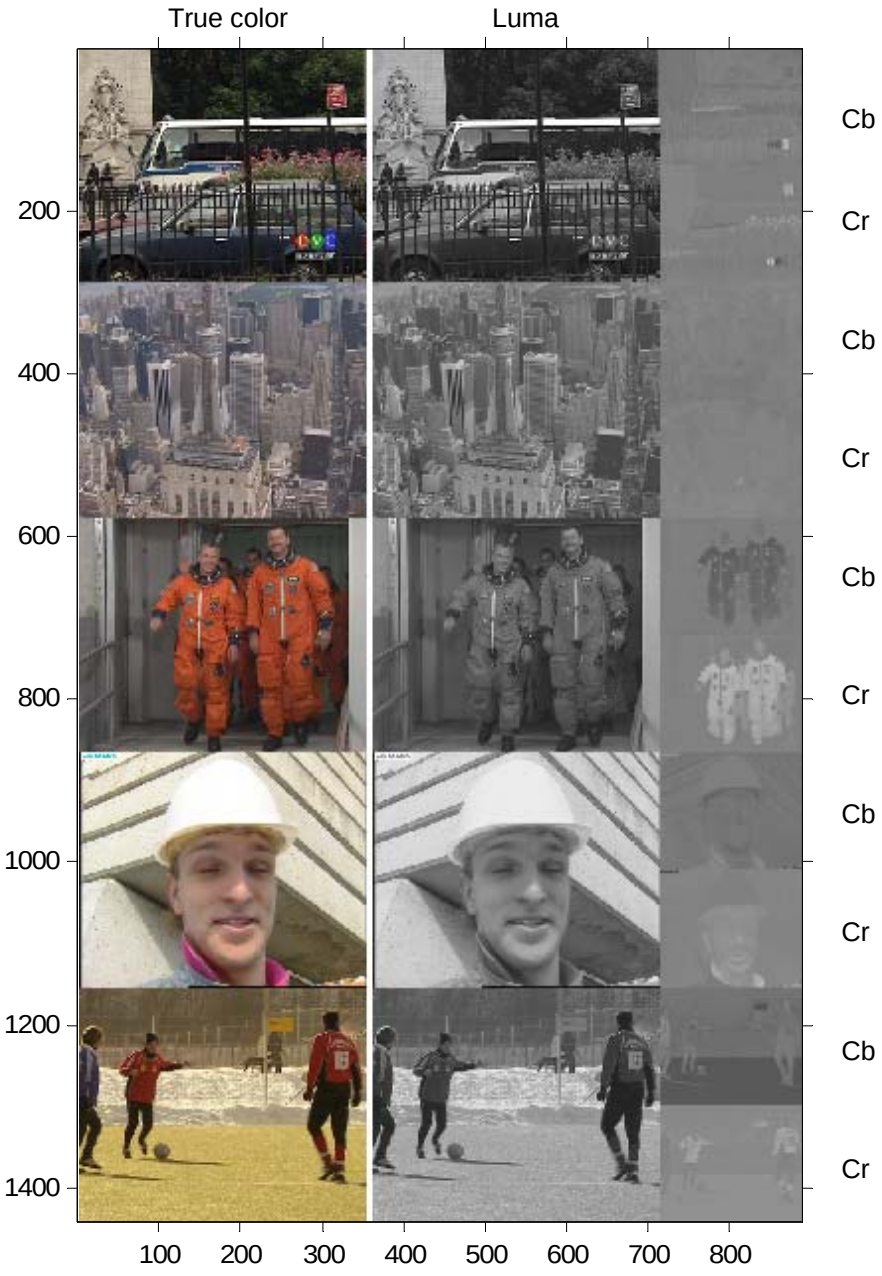
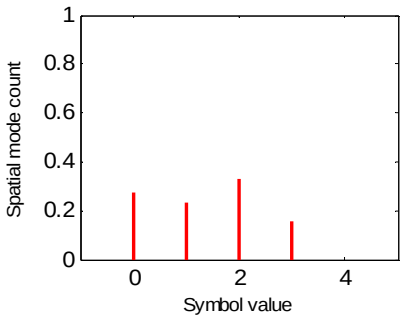


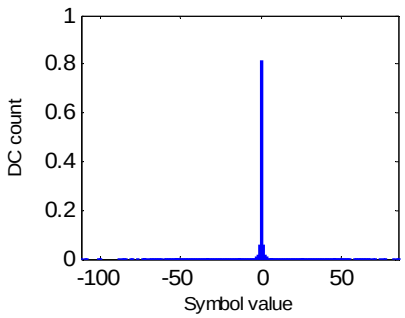
Figure 2.10 Five CIF sequences for codebook training, from top row, Bus frame 20, City frame 40, Crew frame 50, Foreman frame 0, and Soccer frame 0. From left column, show true color, luma, Cb (top), and Cr (bottom) components



Spatial mode codebook ∈ [0 3], size_4 symbols
average length_2 bits, (entropy_1.9532 bits)

0	[0 1]
1	[1 0]
2	[0 0]
3	[1 1]

(a)



DC codebook ∈ [-112 85], size_158 symbols
average length_1.535 bits, (entropy_1.2098 bits)

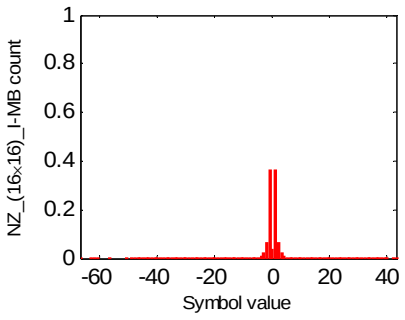
…	…
-4	[1 1 1 1 0 1]
-3	[1 1 0 0 0 0]
-2	[1 1 1 0]
-1	[1 0 1]
0	[0]
1	[1 0 0]
2	[1 1 0 1]
3	[1 1 1 1 1]
4	[1 1 1 1 0 0]
…	…

(b)

Figure 2.11 Histogram and the corresponding Huffman codebook of,
(a) Spatial_mode, (b) dc coefficient

Table 2.4 Huffman Codebook Names and Their Descriptions for I-Frame Coding

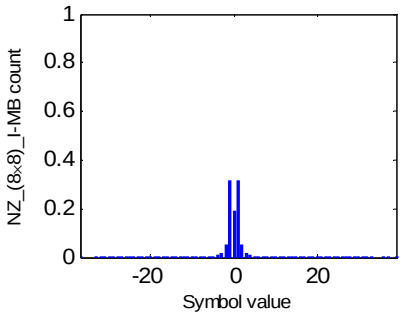
Codebook name	Component type	Description
Spatial_mode	Luma	Spatial prediction modes
dc	Luma & chroma	dc coefficients for all macroblock types
NZ_16x16_I-MB	Luma	Non-zero values for sub-block size 16×16
NZ_8x8_I-MB	Luma & chroma	Non-zero values for sub-block size 8×8
NZ_4x4_I-MB	Luma & chroma	Non-zero values for sub-block size 4×4
NZ_2x2_I-MB	Chroma	Non-zero values for sub-block size 2×2
RUN_16x16_I-MB	Luma	Number of zero values for sub-block size 16×16
RUN_8x8_I-MB	Luma & chroma	Number of zero values for sub-block size 8×8
RUN_4x4_I-MB	Luma & chroma	Number of zero values for sub-block size 4×4
RUN_2x2_I-MB	Chroma	Number of zero values for sub-block size 2×2



NZ_(16×16)_I-MB codebk ∈ [-67 43], size_98
avg length_2.4795 bits, (entropy_2.4098 bits)

…	…
-4	[0 1 0 0 0 1 0]
-3	[0 1 0 0 0 0]
-2	[0 1 1 0]
-1	[0 0]
0	[0 1 0 0 1] EndOfNZ
1	[1]
2	[0 1 0 1]
3	[0 1 1 1 1]
4	[0 1 0 0 0 1 1]
…	…

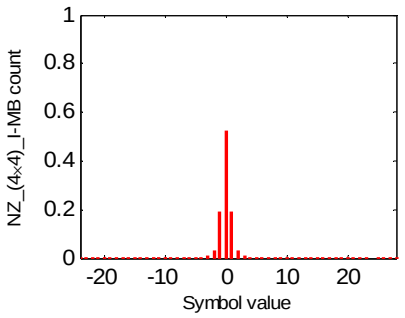
(a)



NZ_(8×8)_I-MB codebk ∈ [-37 39], size_71
avg length_2.5149 bits, (entropy_2.4588 bits)

…	…
-4	[1 1 1 0 0 1]
-3	[1 1 1 1 0]
-2	[1 1 0 1]
-1	[0 1]
0	[1 0] EndOfNZ
1	[0 0]
2	[1 1 0 0]
3	[1 1 1 0 1]
4	[1 1 1 1 1 1]
…	…

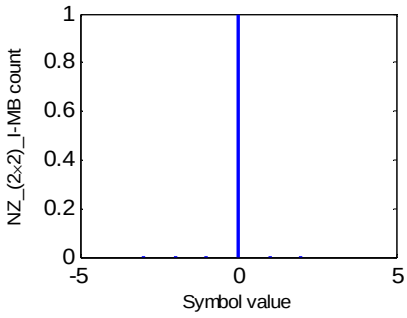
(b)



NZ_(4×4)_I-MB codebk ∈ [-24 28], size_52
avg length_2.0059 bits, (entropy_1.9598 bits)

…	…
-4	[1 0 1 1 1 0 0]
-3	[1 0 1 1 0 0]
-2	[1 0 1 0 0]
-1	[1 0 0]
0	[0] EndOfNZ
1	[1 1]
2	[1 0 1 0 1]
3	[1 0 1 1 0 1]
4	[1 0 1 1 1 1 0]
…	…

(c)

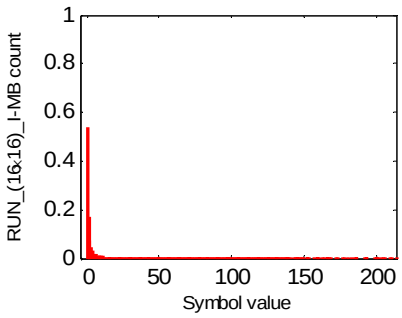


NZ_(2×2)_I-MB codebk ∈ [-3 2], size_6 symbols
avg length_1.0101 bits, (entropy_0.065295 bits)

-3	[1 1 1 1 1]
-2	[1 1 1 1 0]
-1	[1 1 0]
0	[0] EndOfNZ
1	[1 0]
2	[1 1 1 0]

(d)

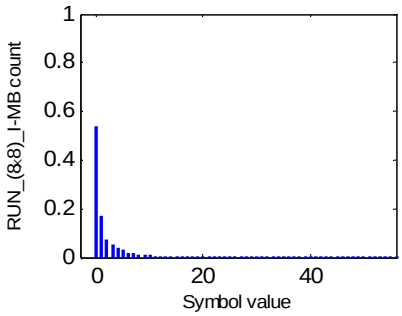
Figure 2.12 Histogram and codebook of NZ, block sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_I-MB codebk ∈ [0 214], size_185
avg length_2.6659 bits, (entropy_2.6294 bits)

0	[0]
1	[1 1]
2	[1 0 0 1]
3	[1 0 0 0 1]
4	[1 0 1 1 1]
5	[1 0 1 0 0 0]
6	[1 0 1 0 1 1]
7	[1 0 1 1 0 1]
8	[1 0 0 0 0 1 0]
…	…

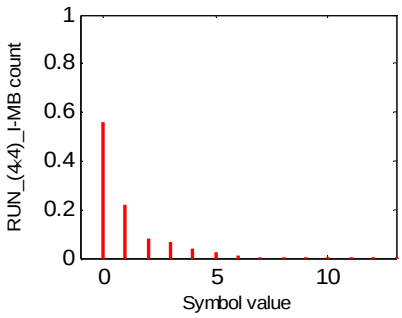
(a)



RUN_(8×8)_I-MB codebk ∈ [0 56], size_57
avg length_2.4339 bits, (entropy_2.3967 bits)

0	[0]
1	[1 1]
2	[1 0 0 1]
3	[1 0 0 0 0]
4	[1 0 1 0 0]
5	[1 0 1 1 0]
6	[1 0 0 0 1 0]
7	[1 0 1 0 1 0]
8	[1 0 1 1 1 0]
…	…

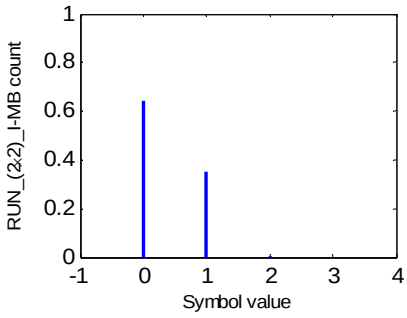
(b)



RUN_(4×4)_I-MB codebk ∈ [0 13], size_14
avg length_1.9487 bits, (entropy_1.9243 bits)

0	[0]
1	[1 1]
2	[1 0 1]
3	[1 0 0 1]
4	[1 0 0 0 0]
5	[1 0 0 0 1 0]
6	[1 0 0 0 1 1 0]
7	[1 0 0 0 1 1 1 1]
8	[1 0 0 0 1 1 1 0 0]
…	…

(c)



RUN_(2×2)_I-MB codebk ∈ [0 2], size_3 symbols
avg length_1.3554 bits, (entropy_0.9644 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

Figure 2.13 Histogram and codebook of RUN, block sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2

CHAPTER 3

INTERFRAME CODING WITH SAD AND SSIM

For interframe coding (IPP···), the proposed codec (Fig. 1.2) uses the following coding tools implemented in MATLAB. Variable size integer transforms are used with {(16×16),(8×8), (4×4)} for the luminance block, and {(8×8),(4×4),(2×2)} for the chrominance block (discussed in Chapter 2). Reference frame interpolation for fractional-pixel motion vector, is used on both luminance and chrominance reference frames similar to H.264. Fast motion estimation (similar to JM H.264 reference software [7][8]) with UMHexagonS motion search for integer pixel resolution and CBFPS motion search for fractional pixel resolution, is used [35][36][37][38]. Huffman entropy encoder is used to obtain the encoded bit stream. The details of each encoding tools are provided in this chapter.

3.1 Reference Frame Interpolation (H.264-Like)

The codec uses quarter-pixel interpolation on luma reference frame and 1/8-pixel interpolation on the chroma reference frame (Fig. 1.2). For a luminance reference frame, half-pixel positions are, first, derived. Then, the quarter-pixel positions are derived. For a chroma reference frame, linear interpolation is applied directly to integer-pixel positions to obtain 1/8-pixel position. Frame boundary reflection is used for the missing pixel interpolation near the frame boundary.

3.1.1 Quarter-pixel interpolation on luminance reference frame

Half-pixel interpolation is performed first. The luma frame is upsampled by 2. Then, convolutions are performed with the half-pixel interpolation filer, $[1,-5,20,20,-5,1]/32$. The interpolation filter is applied along both the row and the column directions [1][4][39]. Then, apply

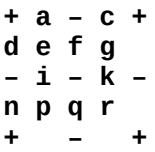


Figure 3.1 Fractional-pixel positions among integer-pixel positions at the four corners. + is an integer-pixel position, – is a half-pixel position, and {a,c,d,e,f,g,i,k,n,p,q,r} are the quarter-pixel positions

rounding and clipping at 255 (maximum pixel value for 8 bpp).

For quarter-pixel interpolation, the bilinear filter, $\left[\frac{1}{2},\frac{1}{2}\right]$, is applied after the half-pixel resolution frame is upsampled by 2 [1][4][39]. Then, the interpolated pixel values are rounded and clipped at 255. Fractional pixels in between integer pixels are shown in Fig. 3.1.

3.1.2 1/8-pixel interpolation on chrominance reference frame

The process for 1/8-pel interpolation is as follows. A chroma reference frame is upsampled by 8 (Fig. 3.2). Frame boundary reflection is used for interpolation at the frame boundary. Then, linear interpolation of neighboring integer pixels is performed in (3.1), [1][4].

$$\mathrm{a} = \mathrm{round}\left(\left[\left(8-d_x\right)\left(8-d_y\right)\mathrm{A} + d_x\left(8-d_y\right)\mathrm{B} + \left(8-d_x\right)d_y\mathrm{C} + d_x d_y\mathrm{D}\right]/64\right), \qquad (3.1)$$

where, ‘a’ is the fractional pixel to be interpolated. {A,B,C,D} are neighboring integer pixels. d_x is pixel distance from the left of the pixel, ‘a’. d_y is pixel distance from the top of the pixel, ‘a’ (in Figure 3.2, d_x is 3, and d_y is 2). Figures 3.3–3.5 show the interpolation results of luma and chroma reference frames for Football test sequence.

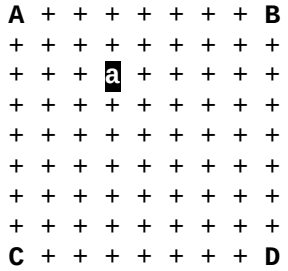
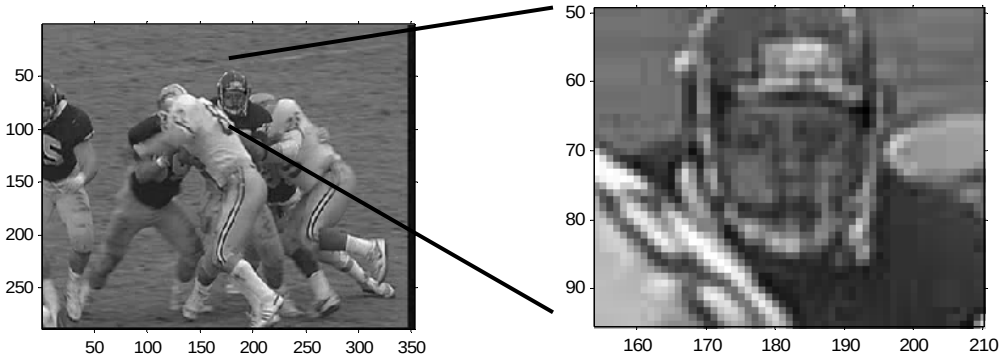
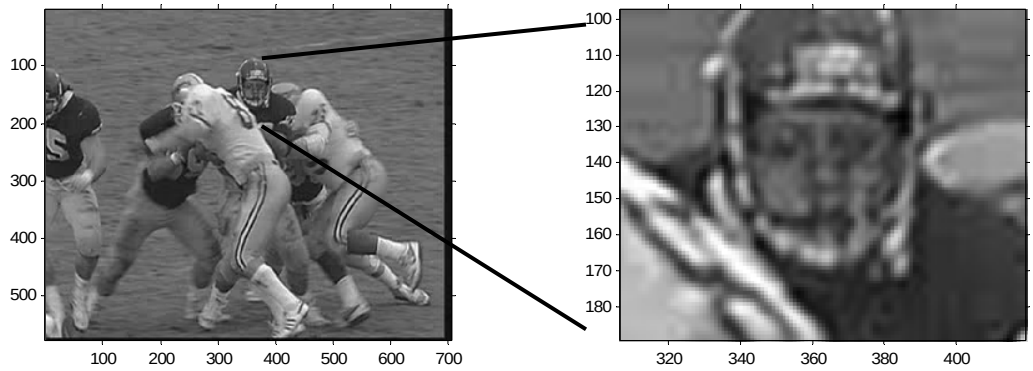


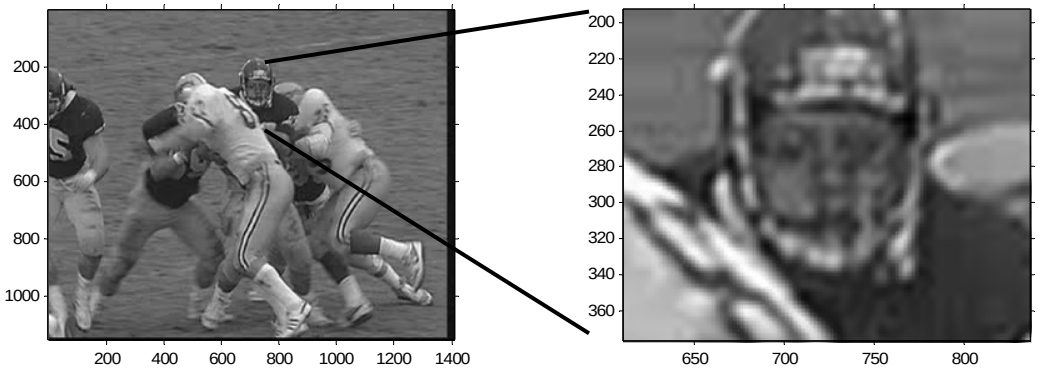
Figure 3.2 Fractional-pixel position of chroma reference frame. {A,B,C,D} are integer-pixel position. + is the pixel with upsampling by 8. ‘a’ is a fractional pixel to be interpolated



(a)

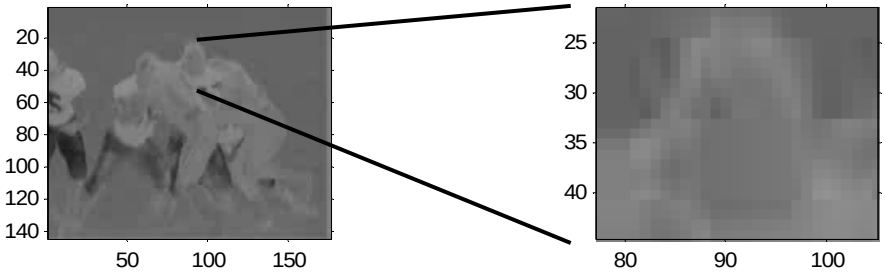


(b)

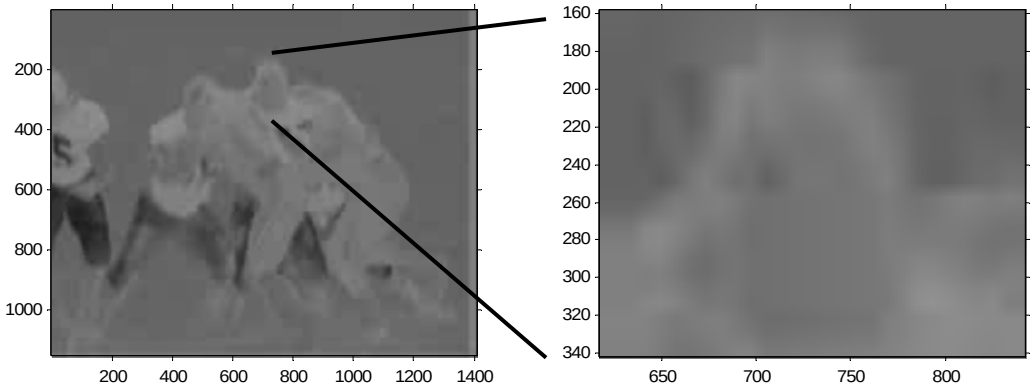


(c)

Figure 3.3 Interpolation on the reconstructed luma frame 50 of Football CIF, *QP* = 30. Left column is full frame, right column is zoomed, (a) reconstructed frame 50, integer-pixel resolution, (b) half-pixel interpolation, (c) quarter-pixel interpolation

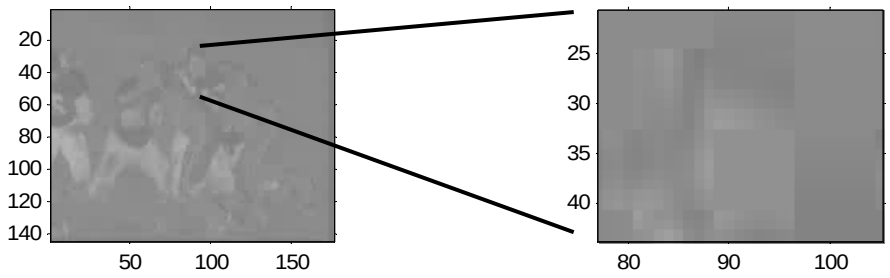


(a)

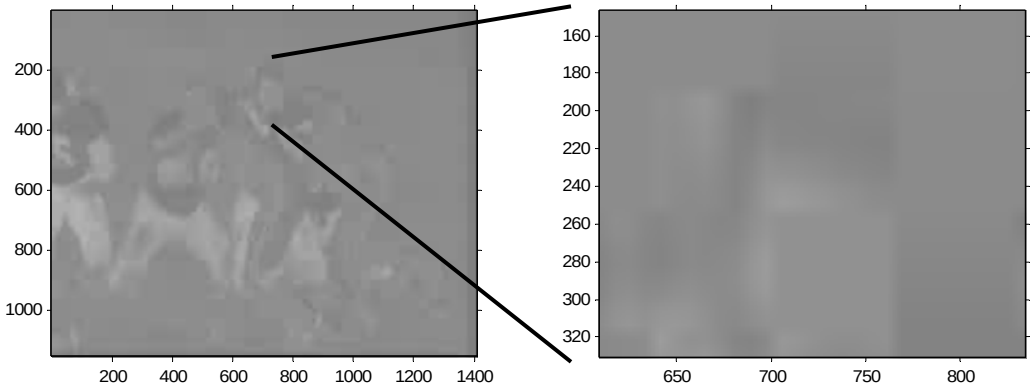


(b)

Figure 3.4 Interpolation on the reconstructed chroma Cb frame 50 of Football CIF, *QP* = 30. Left column is full frame, right column is zoomed, (a) reconstructed frame 50, integer-pixel resolution, (b) 1/8 th-pixel interpolation



(a)



(b)

Figure 3.5 Interpolation on the reconstructed chroma Cr frame 50 of Football CIF, *QP* = 30. Left column is full frame, right column is zoomed, (a) reconstructed frame 50, integer-pixel resolution, (b) 1/8 th-pixel interpolation

3.2 Motion Vector Search Method (JM-Like)

In block-based motion estimation, motion vector is obtained by searching for a reference luma block that gives the best match with the current block within a finite search window in a luma reference frame. Search point is defined as a position of a search block on a reference frame. The search method of the proposed interframe codec is similar to the search method used in the JM reference software (without early termination, for simplicity), called fast motion estimation. The hybrid unsymmetrical-cross multi-hexagon-grid search (UMHexagonS) is used for integer-pixel motion estimation, and the center-biased fractional-pixel search (CBFPS) is used for fractional-pixel motion estimation [32][33][34][35]. Both UMHexagonS and CBFPS are the search methods that combine multiple search techniques to reduce the number of search points compared to a full search. The details of both methods are provided in Sections 3.2.2–3.2.3. Search range of the proposed codec is set at [–32,+32] pixels. With the same search range, full search requires 1,089 searching points.

3.2.1 Initializing motion vector

Before motion search begins, the motion vector mv_{Pred} is predicted from a median of motion vectors of neighboring sub-blocks, mv_{A}, mv_{B}, and mv_{C} in (3.2).

$$mv_{\text{Pred}} = \text{median}\left[\text{round}\left(\left[mv_{\text{A}}, mv_{\text{B}}, mv_{\text{C}}\right]/4\right)\right], \tag{3.2}$$

where, {A, B, C, D} are neighboring sub-blocks. Median is the middle number. Round is to obtain the nearest integer. The divided-by-4 motion vector is to obtain the value at the nearest integer resolution from quarter-pixel resolution. Procedure to obtain the predicted motion vector is shown as the flow chart in Fig. 3.6. mv_{Pred} is in integer-pixel resolution. Examples of neighboring sub-blocks and macroblocks of different sub-block sizes are shown in Fig. 3.7.

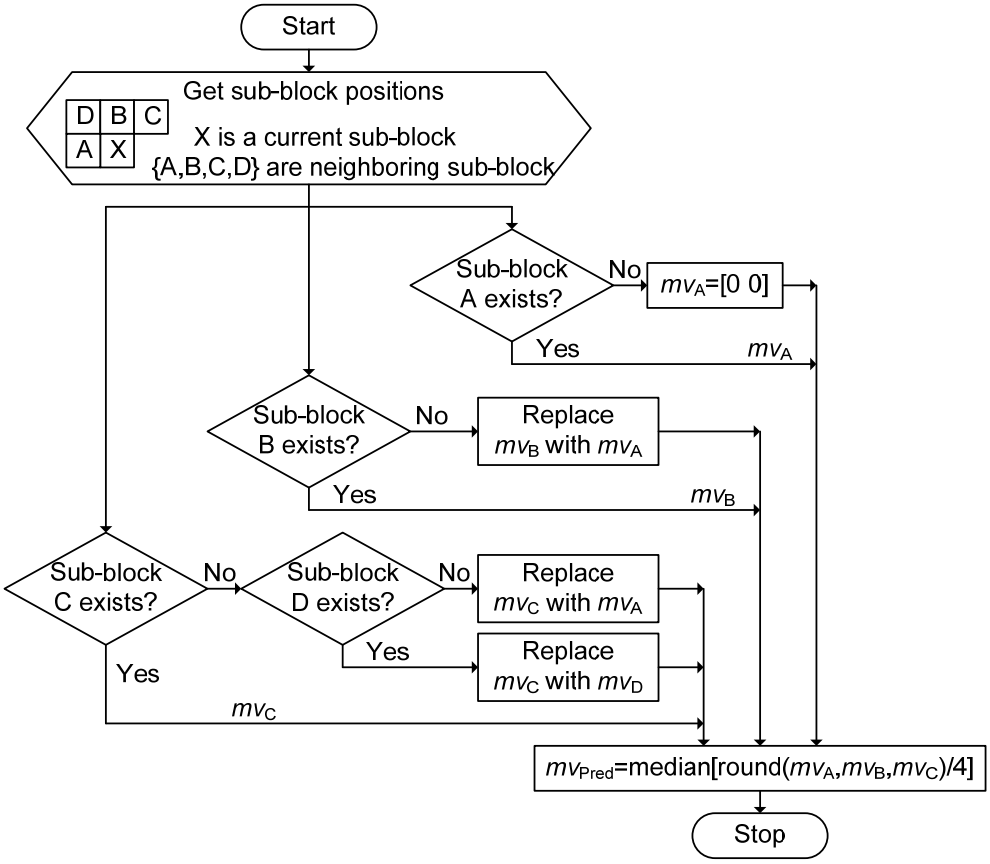
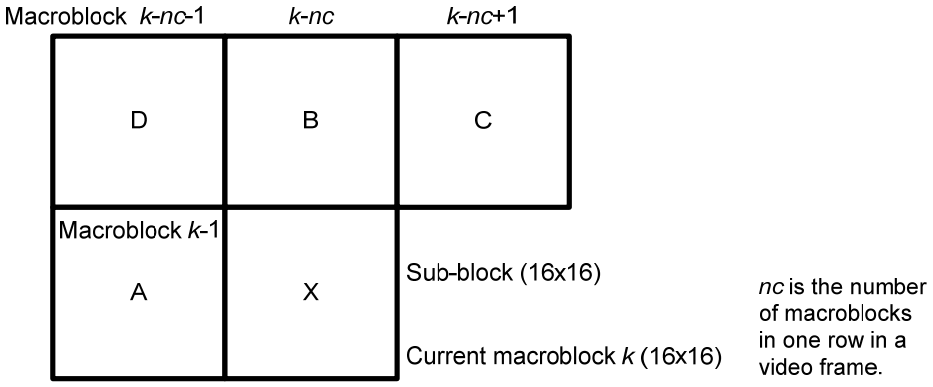
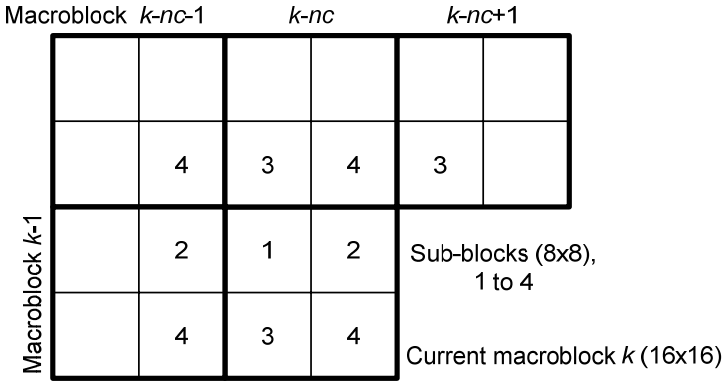


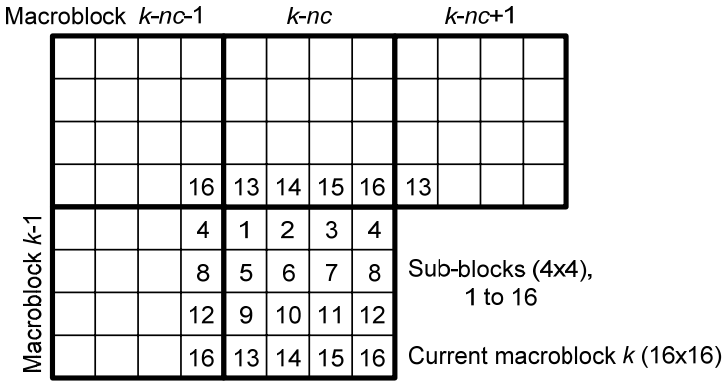
Figure 3.6 Flow chart for predicting the initial motion vector from neighboring motion vectors



(a)



(b)



(c)

Figure 3.7 Neighboring sub-blocks and macroblocks to predict the initial motion vector of different sub-block sizes, (a) (16×16), (b) (8×8), (c) (4×4)

3.2.2 Integer-pixel search for motion vector

The search procedure for integer-pixel motion vector is as follows. Figure 3.8 shows the flow chart for integer-pixel search.

Step 1: Initializing the motion vector search. Select either motion vector $(0,0)$ (the same position as the current block) or mv_{Pred} (median of motion vectors from neighboring macroblocks).

Step 2: Asymmetrical cross search consists of 16 horizontal and 8 vertical search points, excluding the center position (Figure 3.9(a)) for motion block sizes (16×16) and (8×8).

Step 3: Uneven multi-hexagon-grid search (Figure 3.9(b)) for motion block sizes (16×16) and (8×8), consists of two steps,

Step 3.1: Small full local search covers [±2, ±2] pixels (24 search points).

Step 3.2: Uneven-16-point-multi-hexagon-grid search consists of four hexagon search (64 search points).

Step 4: Extended hexagon-based search (EHS) (Figure 3.9(c)) for all motion block sizes (16×16), (8×8), and (4×4), consists of two steps,

Step 4.1: Hexagon based search (HEXBS) [40] consists of 6 search points with 3 more search points on every extended search.

Step 4.2: Diamond search (DS) [41] or small hexagon search, consists of 4 search points.

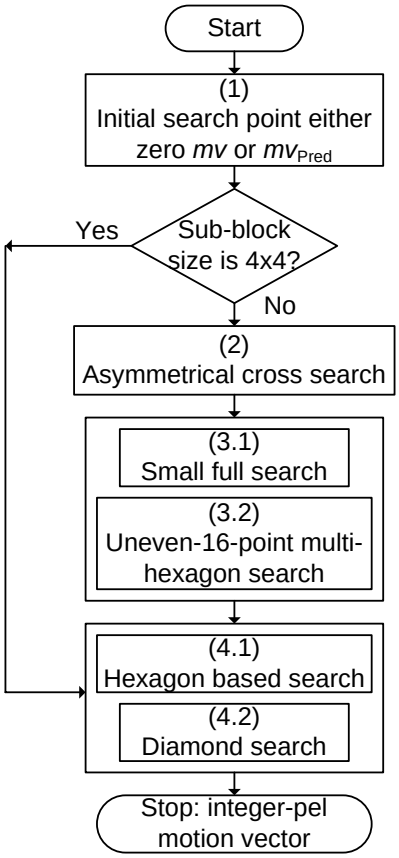
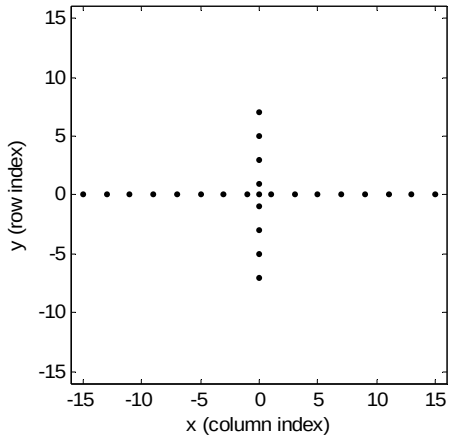
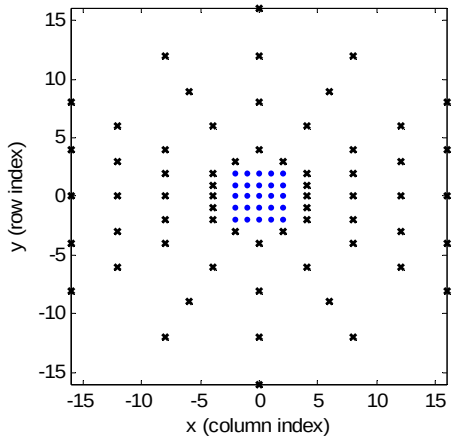


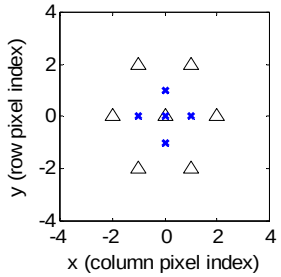
Figure 3.8 Flow chart of integer-pixel search [37] with modification



(a)



(b)



(c)

Figure 3.9 Hybrid unsymmetrical-cross multi-hexagon-grid search pattern on an integer-pixel luma reference frame. (a) asymmetrical cross search, (b) uneven multi-hexagon-grid search, (c) extended hexagon-based search (EHS)

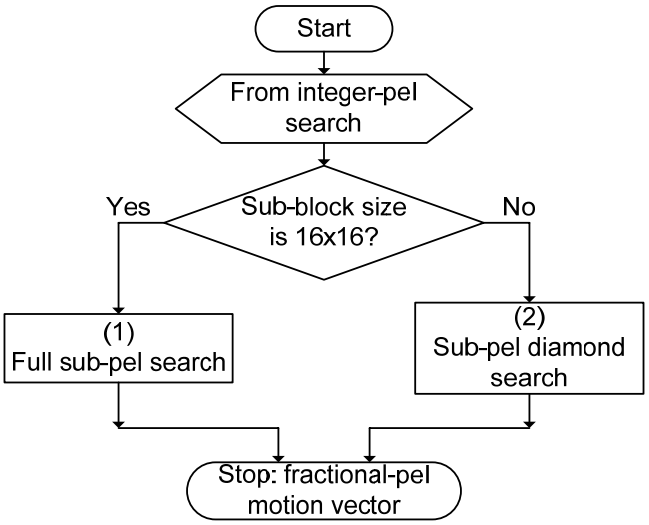


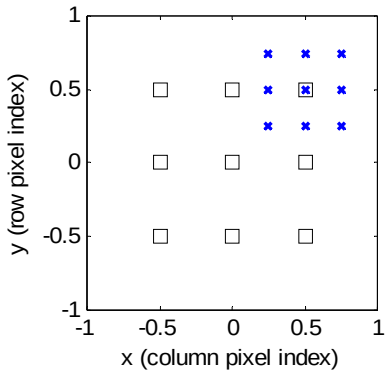
Figure 3.10 Flow chart of fractional-pixel search [37] with modification

3.2.3 Fractional-pixel search for motion vector

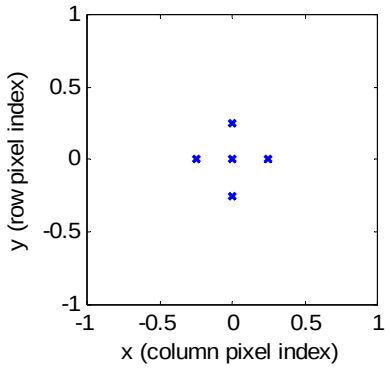
A flow chart of the fractional-pixel search is shown in Figure 3.10. Fractional-pixel search [35] occurs as follows.

Step 1: Full sub-pixel search for motion block size (16×16), consists of 16 search points (Figure 3.11(a)).

Step 2: Diamond search (called fractional-pixel fast search) for motion block sizes (8×8), and (4×4), consists of 4 search points with 3 more search points on every extended search (Figure 3.11(b)).



(a)



(b)

Figure 3.11 Center-biased fractional pixel search pattern on a fractional-pixel interpolated luma reference frame. (a) fractional-pixel full search, (b) fractional-pixel fast search, diamond search

3.3 Block Matching Using SAD and SSIM

This section provides the concept of block matching in block-based motion estimation. A motion vector is obtained by searching a reference luma block of a reference frame that gives the best match with the current block of a current frame (Fig. 3.12) within a search window. Best matching is defined as the minimum distortion among the measurements. In the codec simulation, motion vector, mv_{luma}, is estimated on a luma block. The motion vector of the corresponding chroma block, mv_{chroma}, is derived from the luma motion vector in (3.3) for the 4:2:0 sampling format of video sequence.

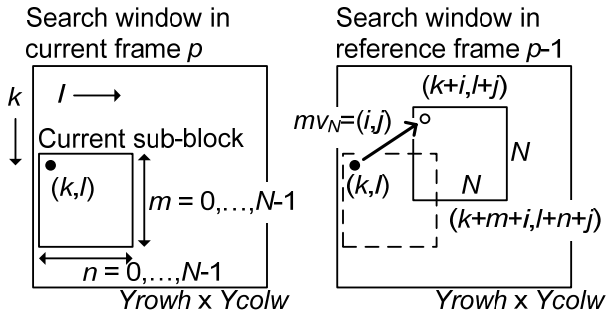


Figure 3.12 Block-matching motion prediction

$$mv_{\text{chroma}} = \frac{mv_{\text{luma}}}{2}, \tag{3.3}$$

The concept of block-based motion estimation can be found in [42]. An example of motion compensated frame for (16×16) motion block prediction of Football sequence is shown in Fig. 3.13. The research considers two types of distortions, the conventional sum of absolute difference (SAD), and structural similarity (SSIM).

3.3.1 SAD distortion calculation

SAD is widely used in the block-based codec. SAD of each block matching is calculated in (3.4),

$$D_N^{\text{SAD}}(k,l;i,j) = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1}\left|y(k+m,l+n,p) - y_{\text{Ref}}(k+m+i,l+n+j,p-1)\right|, \tag{3.4}$$

where, $D_N^{\text{SAD}}(k,l;i,j)$ is the SAD distortion value between a current block at (k,l) position and a reference block at $(k+i,l+j)$ position, with a size of (N×N). The position of a block is determined by the top-left corner pixel location. The displacement vector of a reference block is determined by (i,j). p and $p-1$ indicate the frame number. MATLAB function used to calculate the SAD value is listed in Appendix B.

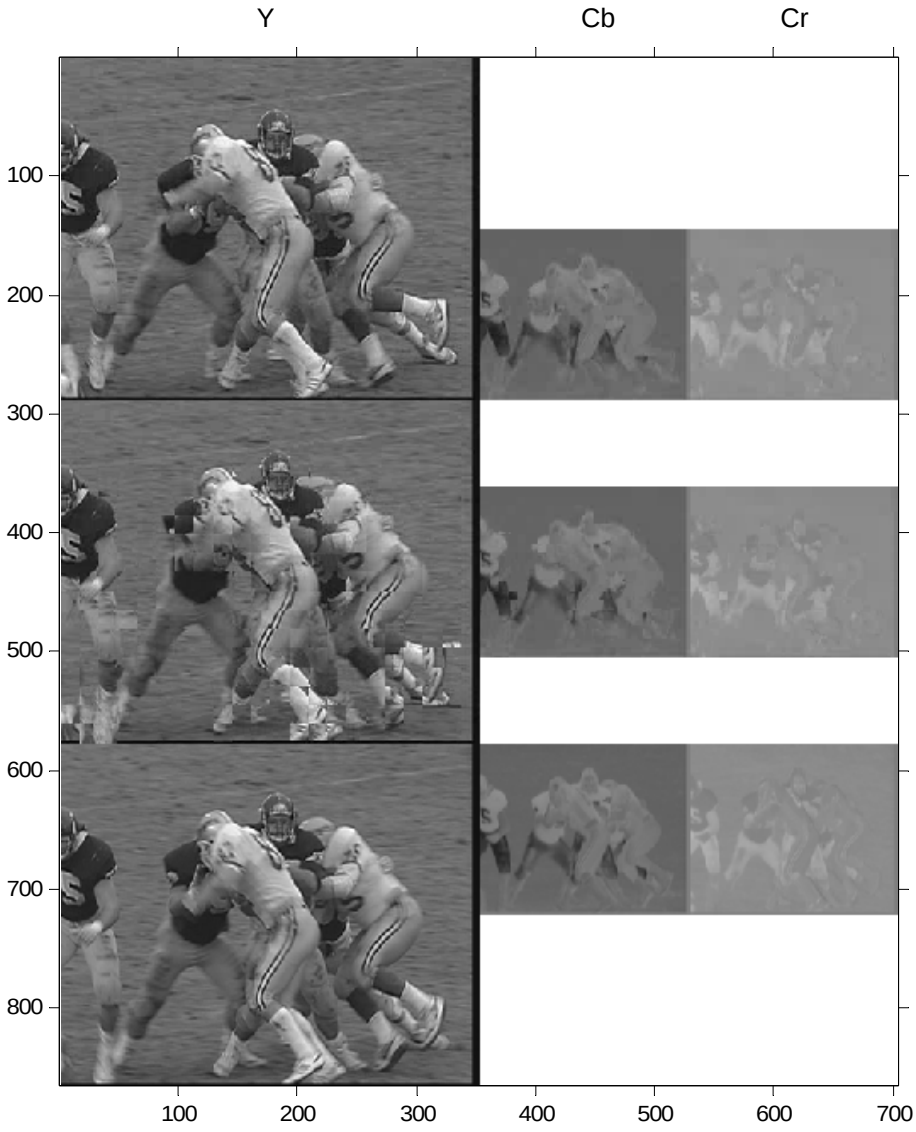


Figure 3.13 An example of motion estimation on Football CIF 4:2:0, *QP* = 30, motion prediction with SSIM-based distortion, fixed-(16×16) motion block. From top row, reconstructed reference frame 50, motion-predicted frame 51, and current frame 51. From left column, Y, Cb, and Cr components

A motion vector, $mv_N(k,l)$ of a current block is the displacement vector (i_1, j_1) with the minimum distortion value $D_N(k,l)$, in (3.5)–(3.6). In the simulation, motion block size *N* is 4, 8, and 16.

$$D_N(k,l) = \min_{(i,j)} \left[D_N^{\text{SAD}}(k,l;i,j) \right], \tag{3.5}$$

$$mv_N(k,l) = (i_1, j_1), \tag{3.6}$$

3.3.2 SSIM distortion calculation

For SSIM distortion, the distortion is defined in (3.7). The minimum distortion is 0 and the maximum distortion is 1,

$$D_N^{\text{SSIM}}(k,l;i,j) = 1 - \text{SSIM}, \tag{3.7}$$

A motion vector, $mv_N(k,l)$ of a current block is the displacement vector (i_2, j_2) with the minimum distortion value $D_N(k,l)$, in (3.8)–(3.9).

$$D_N(k,l) = \min_{(i,j)} \left[D_N^{\text{SSIM}}(k,l;i,j) \right], \tag{3.8}$$

$$mv_N(k,l) = (i_2, j_2), \tag{3.9}$$

In block-matching motion estimation, the calculation of SSIM and the parameter setting are based on (1.8)–(1.9). The research uses a local window size same as a sub-block size, without moving the local window. For sub-block size of (16×16), the research uses (16×16)-local window. For sub-block size of (8×8), the research uses (8×8)-local window. For sub-block size of (4×4), the research uses (4×4)-local window. MATLAB implementation of SSIM distortion calculation is provided in Appendix B.

3.4 Bit Stream Format

Output from the transform encoding of the residual block is mapped into bits by the Huffman entropy encoder. The proposed bit stream format for interframe encoding is given in Table 3.1. **`coeffBSS`** (block size selection bits), **`dc`** (composite dc coefficients), **`NZ`** (non-zero

Table 3.1 The Bit Stream Format of the Proposed Codec for a Macroblock of P-Frame

Macroblock type	Bit stream format
Luma P-MB	$\mathbf{mv_{V1} \cdots mv_{Vq}}$, $\mathbf{mv_{H1} \cdots mv_{Hq}}$, **coeffBSS,** $\mathbf{dc_1 \cdots dc_n}$, $\mathbf{NZ_1\ RUN_1 \cdots NZ_i\ RUN_i}$ **EndOfNZ,** … $\mathbf{NZ_1\ RUN_1 \cdots NZ_j\ RUN_j}$ **EndOfNZ**
Chroma P-MB	**coeffBSS,** $\mathbf{dc_1 \cdots dc_m}$, $\mathbf{NZ_1\ RUN_1 \cdots NZ_k\ RUN_k}$ **EndOfNZ,** … $\mathbf{NZ_1\ RUN_1 \cdots NZ_p\ RUN_p}$ **EndOfNZ**

values from run-length encoder), **RUN** (number of zero values from RLC), and **EndOfNZ** are the coefficients from the transform encoding block. End of non-zero symbol **EndOfNZ** (bit '0') in the bit stream indicates the end of each sub-block. The bit stream format objective is to ensure that the bit stream can be decoded correctly.

3.5 Codebook Training of P-Frame

In the simulations, Huffman codebooks are trained with five test sequences (Bus, City (ABC), Crew (NASA), Foreman, and Soccer) (Fig. 2.10). Video test sequences (in YUV raw format) can be obtained from [33]. Motion vectors, dc coefficients and coefficients of different transform sizes from run-length code (RLC) are collected. Histograms for all coefficient types are created. Huffman codebooks are, then, obtained from Huffman trees of the histograms. The Huffman tree method can be found in [34]. A list of codebook names of the simulated codec is given in Table 3.2. For each motion block size, two codebook sets are created based on distortion calculation methods. This is done using a SAD-based codebook (motion prediction based on SAD), and a SSIM-based codebook (motion prediction based on SSIM distortion). Figures 3.14–3.31 show Huffman codebooks of different coefficients including the histogram,

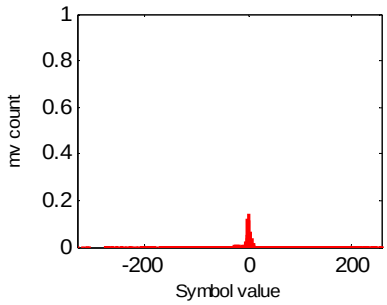
codebook range, size of the codebook, average code length and its entropy in bits per symbol, for SAD-based and SSIM-based codebooks.

Histograms of motion vector between SAD-based and SSIM-based are similar in shape and range (Figs. 3.14(a) and 3.23(a), Figs. 3.17(a) and 3.26(a), Figs. 3.20(a) and 3.29(a)). Histograms of composite dc coefficient between SAD-based and SSIM-based have a similar shape however SAD-based has a smaller range (Figs. 3.14(b) and 3.23(b), Figs. 3.17(b) and 3.26(b), Figs. 3.20(b) and 3.29(b)).

Histograms of NZ between SAD-based and SSIM-based have a similar shape. However, SAD-based has a smaller range (Figs. 3.15 and 3.24, Figs. 3.18 and 3.27, Figs. 3.21 and 3.30). Histograms of RUN between SAD-based and SSIM-based are similar both in shape and range (Figs. 3.16 and 3.25, Figs. 3.19 and 3.28, Figs. 3.22 and 3.31).

Table 3.2 Huffman Codebook Names and Their Descriptions for P-Frame Coding

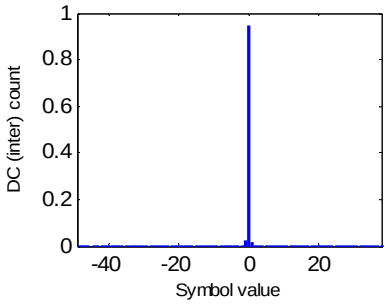
Codebook name	Component type	Description
mv	Luma	Motion vector components both vertical (column) and horizontal (row) directions
dc	Luma & chroma	dc coefficients for all macroblock types
NZ_16x16_P-MB	Luma	Non-zero values for sub-block size 16×16
NZ_8x8_P-MB	Luma & chroma	Non-zero values for sub-block size 8×8
NZ_4x4_P-MB	Luma & chroma	Non-zero values for sub-block size 4×4
NZ_2x2_P-MB	Chroma	Non-zero values for sub-block size 2×2
RUN_16x16_P-MB	Luma	Number of zero values for sub-block size 16×16
RUN_8x8_P-MB	Luma & chroma	Number of zero values for sub-block size 8×8
RUN_4x4_P-MB	Luma & chroma	Number of zero values for sub-block size 4×4
RUN_2x2_P-MB	Chroma	Number of zero values for sub-block size 2×2



mv codebook ∈ [-331 257] size_438 symbols
avg length_4.8872 bits, (entropy_4.8673 bits)

…	…
-4	[1 0 0 1 1 0]
-3	[1 1 0 0]
-2	[0 1 0 0 0]
-1	[0 1 1]
0	[0 0 1]
1	[1 0 1]
2	[1 1 1 0]
3	[1 0 0 0]
4	[0 0 0 0 1 0]
…	…

(a)

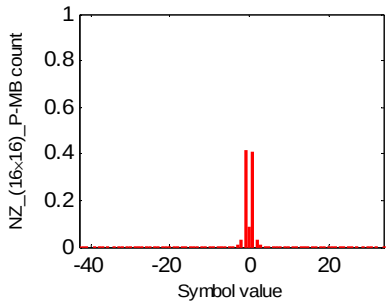


DC codebook ∈ [-49 38] size_87 symbols
avg length_1.0985 bits, (entropy_0.37849 bits)

…	…
-4	[1 0 1 1 1 1 1]
-3	[1 0 1 1 1 0]
-2	[1 0 1 0 0]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 0 1]
3	[1 0 1 1 0 1]
4	[1 0 1 1 1 1 0]
…	…

(b)

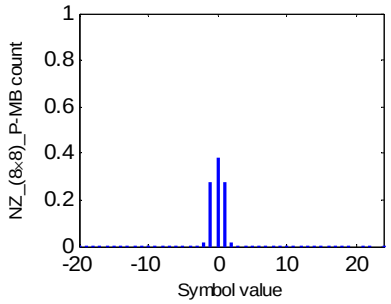
Figure 3.14 Histograms and the corresponding Huffman codebooks for motion block size 16×16 (with SAD-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-43 34] size_74
avg length_1.9804 bits, (entropy_1.8798 bits)

…	…
-4	[0 1 0 0 1 1 0 1]
-3	[0 1 0 0 1 0 0]
-2	[0 1 0 1]
-1	[1]
0	[0 1 1] EndOfNZ
1	[0 0]
2	[0 1 0 0 0]
3	[0 1 0 0 1 0 1]
4	[0 1 0 0 1 1 1 0]
…	…

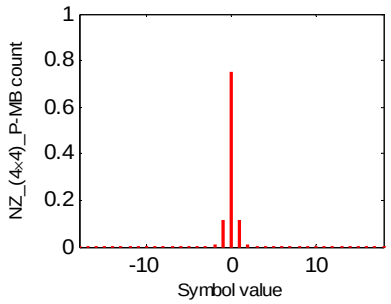
(a)



NZ_(8×8)_P-MB codebook ∈ [-20 24] size_43
avg length_2.1099 bits, (entropy_1.9499 bits)

…	…
-4	[0 0 1 0 1 1 0 1]
-3	[0 0 1 0 1 0 0]
-2	[0 0 1 1]
-1	[0 1]
0	[1] EndOfNZ
1	[0 0 0]
2	[0 0 1 0 0]
3	[0 0 1 0 1 0 1]
4	[0 0 1 0 1 1 1 0]
…	…

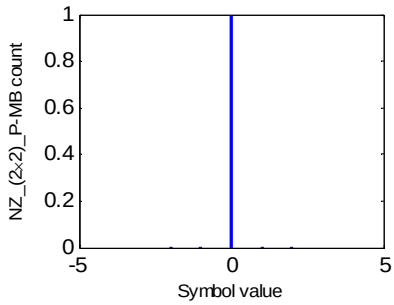
(b)



NZ_(4×4)_P-MB codebook ∈ [-18 18] size_37
avg length_1.446 bits, (entropy_1.2084 bits)

…	…
-4	[1 0 1 0 1 1 0 0]
-3	[1 0 1 0 1 0 0]
-2	[1 0 1 1]
-1	[1 0 0]
0	[0] EndOfNZ
1	[1 1]
2	[1 0 1 0 0]
3	[1 0 1 0 1 0 1]
4	[1 0 1 0 1 1 0 1]
…	…

(c)

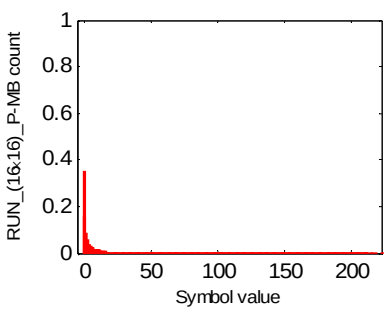


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0012 bits, (entropy_0.0098911 bits)

-2	[1 1 1 1]
-1	[1 0]
0	[0] EndOfNZ
1	[1 1 0]
2	[1 1 1 0]

(d)

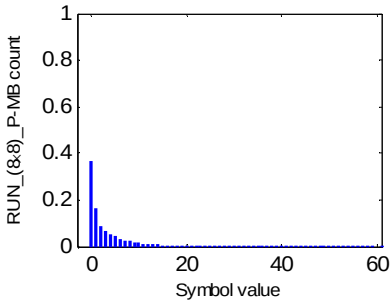
Figure 3.15 Histograms and codebooks (SAD-based) for motion block size 16×16 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 222] size_217
avg length_3.808 bits, (entropy_3.7395 bits)

0	[0 0]
1	[0 1 0]
2	[1 1 0]
3	[1 0 0 0]
4	[1 0 1 1]
5	[0 1 1 0 1]
6	[1 0 0 1 0]
7	[1 0 1 0 1]
8	[1 1 1 0 1]
…	…

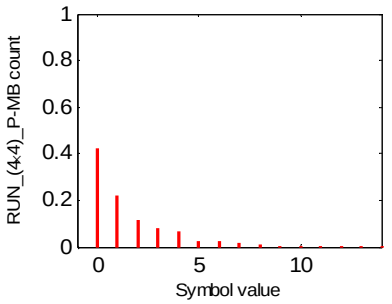
(a)



RUN_(8×8)_P-MB codebook ∈ [0 61] size_61
avg length_3.3959 bits, (entropy_3.3289 bits)

0	[1]
1	[0 0 1]
2	[0 0 0 1]
3	[0 1 1 0]
4	[0 0 0 0 0]
5	[0 1 0 0 0]
6	[0 1 0 1 1]
7	[0 1 1 1 1]
8	[0 0 0 0 1 0]
…	…

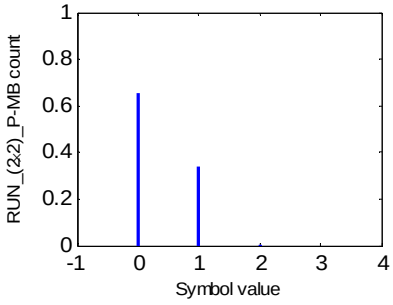
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.531 bits, (entropy_2.4747 bits)

0	[1]
1	[0 1]
2	[0 0 0 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 0 0 1 0 0]
6	[0 0 0 1 0 1]
7	[0 0 0 1 1 1]
8	[0 0 0 1 1 0 1]
…	…

(c)

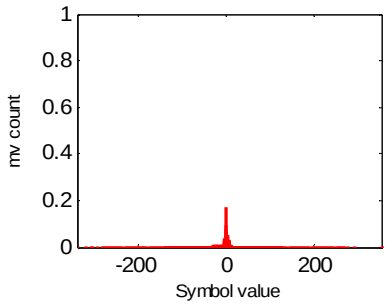


RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.3442 bits, (entropy_0.97286 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

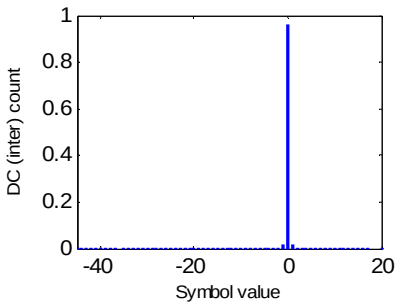
Figure 3.16 Histograms and codebooks (SAD-based) for motion block size 16×16 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



mv codebook ∈ [-340 356] size_342 symbols
avg length_5.3015 bits, (entropy_5.2634 bits)

…	…
-4	[0 0 1 1 0]
-3	[0 1 1 0 0]
-2	[1 0 0 1 1]
-1	[1 1 1]
0	[0 0 0]
1	[0 1 0 0]
2	[0 1 1 1 0]
3	[1 0 1 1]
4	[1 1 0 1]
…	…

(a)

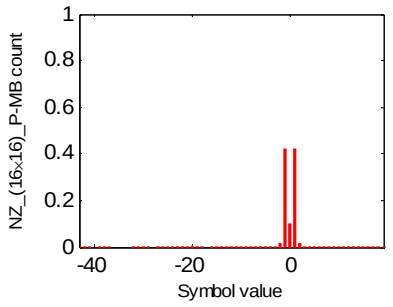


DC codebook ∈ [-45 20] size_63 symbols
avg length_1.0665 bits, (entropy_0.29162 bits)

…	…
-4	[1 0 1 0 1 1 1 0]
-3	[1 0 1 0 1 0 1]
-2	[1 0 1 0 0]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 1]
3	[1 0 1 0 1 0 0]
4	[1 0 1 0 1 1 0 0]
…	…

(b)

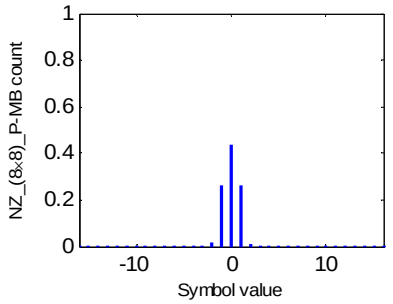
Figure 3.17 Histograms and the corresponding Huffman codebooks for motion block size 8×8 (with SAD-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-43 19] size_56
avg length_1.8317 bits, (entropy_1.6999 bits)

…	…
-4	[0 1 1 0 1 0 1 0 0]
-3	[0 1 1 0 1 1]
-2	[0 1 1 1]
-1	[0 0]
0	[0 1 0] EndOfNZ
1	[1]
2	[0 1 1 0 0]
3	[0 1 1 0 1 0 0]
4	[0 1 1 0 1 0 1 0 1]
…	…

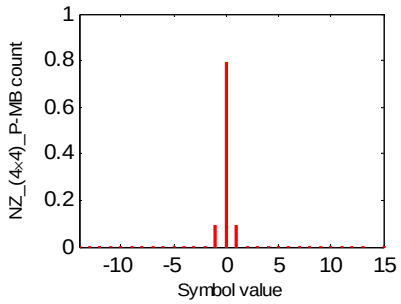
(a)



NZ_(8×8)_P-MB codebook ∈ [-16 16] size_33
avg length_1.9444 bits, (entropy_1.792 bits)

…	…
-4	[0 0 1 0 1 1 0 0]
-3	[0 0 1 0 1 0 0]
-2	[0 0 1 1]
-1	[0 0 0]
0	[1] EndOfNZ
1	[0 1]
2	[0 0 1 0 0]
3	[0 0 1 0 1 0 1]
4	[0 0 1 0 1 1 0 1]
…	…

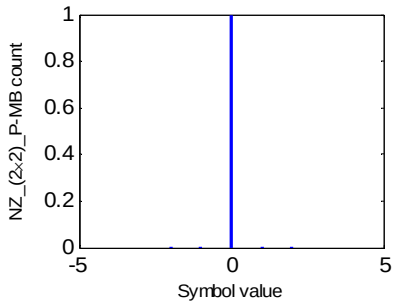
(b)



NZ_(4×4)_P-MB codebook ∈ [-14 15] size_29
avg length_1.3509 bits, (entropy_1.0371 bits)

…	…
-4	[1 0 1 0 1 1 0 0]
-3	[1 0 1 0 1 0 0]
-2	[1 0 1 1]
-1	[1 0 0]
0	[0] EndOfNZ
1	[1 1]
2	[1 0 1 0 0]
3	[1 0 1 0 1 0 1]
4	[1 0 1 0 1 1 0 1]
…	…

(c)

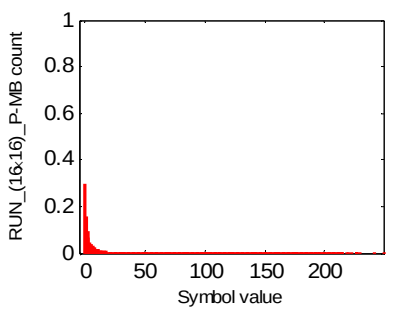


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0012 bits, (entropy_0.010578 bits)

-2	[1 1 1 1]
-1	[1 1 0]
0	[0] EndOfNZ
1	[1 0]
2	[1 1 1 0]

(d)

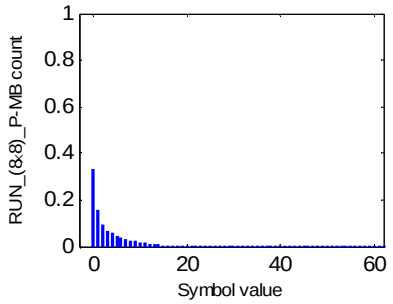
Figure 3.18 Histograms and codebooks (SAD-based) for motion block size 8×8 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 249] size_223
avg length_4.0105 bits, (entropy_3.976 bits)

0	[0 1]
1	[0 0 0]
2	[1 1 0]
3	[0 0 1 1]
4	[1 0 1 1]
5	[0 0 1 0 1]
6	[1 0 0 1 0]
7	[1 0 1 0 1]
8	[1 1 1 0 1]
…	…

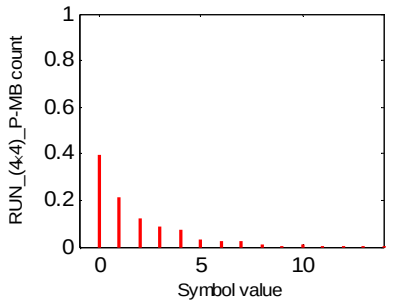
(a)



RUN_(8×8)_P-MB codebook ∈ [0 62] size_63
avg length_3.5575 bits, (entropy_3.5034 bits)

0	[0 0]
1	[0 1 0]
2	[1 1 0]
3	[0 1 1 0]
4	[1 0 0 0]
5	[1 0 1 1]
6	[1 1 1 1]
7	[0 1 1 1 1]
8	[1 0 0 1 1]
…	…

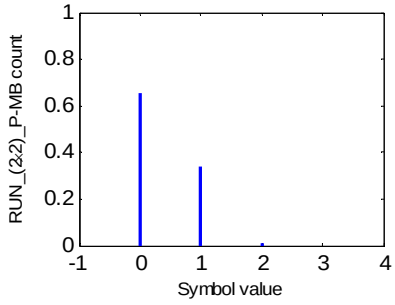
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.6399 bits, (entropy_2.5753 bits)

0	[1]
1	[0 0 0]
2	[0 1 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 1 1 0 0]
6	[0 1 1 1 0]
7	[0 1 1 1 1]
8	[0 1 1 0 1 1]
…	…

(c)

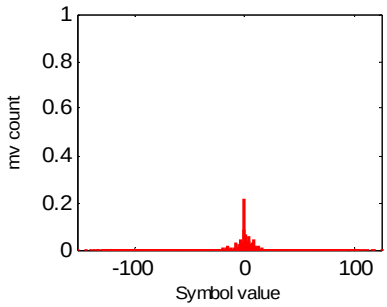


RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.3467 bits, (entropy_0.99553 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

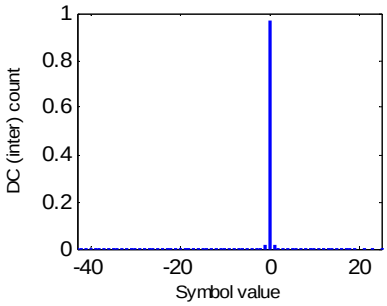
Figure 3.19 Histograms and codebooks (SAD-based) for motion block size 8×8 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



mv codebook ∈ [-152 124] size_244 symbols
avg length_4.7574 bits, (entropy_4.7264 bits)

…	…
-4	[1 1 1 1]
-3	[1 1 1 0 1]
-2	[0 1 1 0 1]
-1	[0 0 0 1]
0	[1 0]
1	[0 1 0 0]
2	[1 1 0 0 1]
3	[0 0 1 1 0]
4	[0 1 1 1]
…	…

(a)

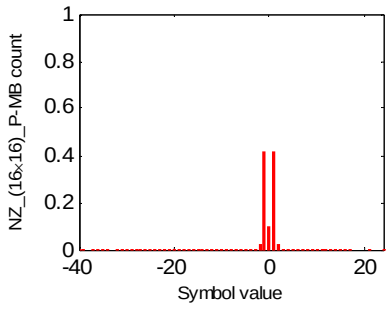


DC codebook ∈ [-43 25] size_66 symbols
avg length_1.0632 bits, (entropy_0.26982 bits)

…	…
-4	[1 0 1 1 1 1 0]
-3	[1 0 1 1 0 1]
-2	[1 0 1 0 1]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 0 0]
3	[1 0 1 1 0 0]
4	[1 0 1 1 1 0 1]
…	…

(b)

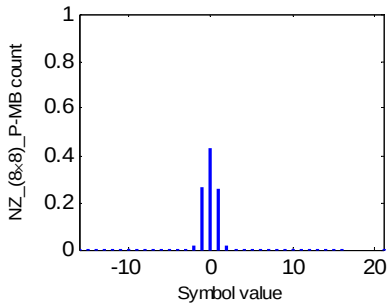
Figure 3.20 Histograms and the corresponding Huffman codebooks for motion block size 4×4 (with SAD-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-40 24] size_58
avg length_1.8902 bits, (entropy_1.7782 bits)

…	…
-4	[0 1 1 0 1 1 0 1]
-3	[0 1 1 0 1 0 0]
-2	[0 1 1 1]
-1	[1]
0	[0 1 0] EndOfNZ
1	[0 0]
2	[0 1 1 0 0]
3	[0 1 1 0 1 0 1]
4	[0 1 1 0 1 1 0 0]
…	…

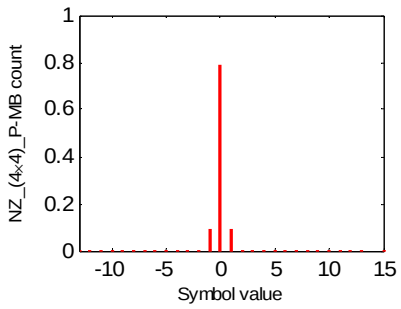
(a)



NZ_(8×8)_P-MB codebook ∈ [-16 21] size_34
avg length_1.9731 bits, (entropy_1.8314 bits)

…	…
-4	[0 0 1 0 1 1 0 1]
-3	[0 0 1 0 1 0 0]
-2	[0 0 1 1]
-1	[0 1]
0	[1] EndOfNZ
1	[0 0 0]
2	[0 0 1 0 0]
3	[0 0 1 0 1 0 1]
4	[0 0 1 0 1 1 0 0]
…	…

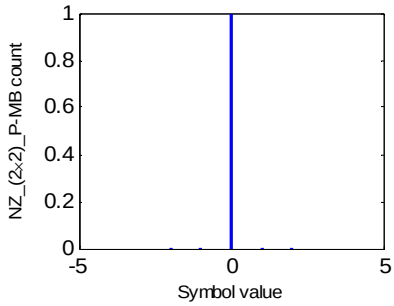
(b)



NZ_(4×4)_P-MB codebook ∈ [-13 15] size_28
avg length_1.3627 bits, (entropy_1.0579 bits)

…	…
-4	[1 0 1 0 1 1 0 1]
-3	[1 0 1 0 1 0 0]
-2	[1 0 1 1]
-1	[1 1]
0	[0] EndOfNZ
1	[1 0 0]
2	[1 0 1 0 0]
3	[1 0 1 0 1 0 1]
4	[1 0 1 0 1 1 0 0]
…	…

(c)

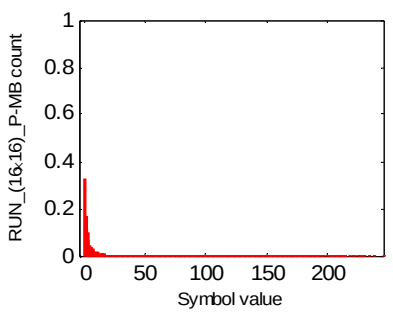


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0023 bits, (entropy_0.018393 bits)

-2	[1 1 1 0]
-1	[1 0]
0	[0] EndOfNZ
1	[1 1 0]
2	[1 1 1 1]

(d)

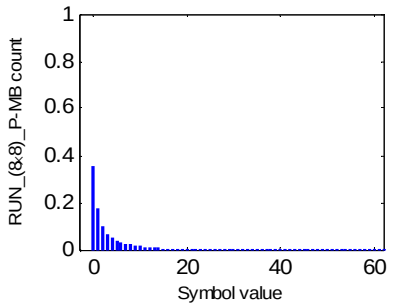
Figure 3.21 Histograms and codebooks (SAD-based) for motion block size 4×4 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 247] size_225
avg length_3.8439 bits, (entropy_3.7883 bits)

0	[0 0]
1	[0 1 0]
2	[1 0 1]
3	[0 1 1 1]
4	[1 1 0 0]
5	[0 1 1 0 0]
6	[1 0 0 0 1]
7	[1 1 0 1 0]
8	[1 1 1 0 1]
…	…

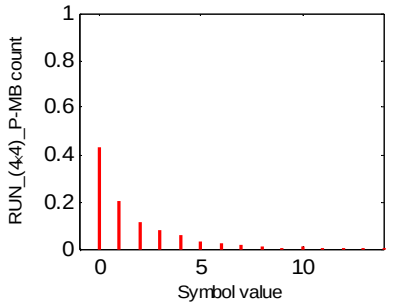
(a)



RUN_(8×8)_P-MB codebook ∈ [0 62] size_63
avg length_3.4154 bits, (entropy_3.3555 bits)

0	[0 0]
1	[1 1]
2	[1 0 1]
3	[0 1 1 0]
4	[1 0 0 0]
5	[0 1 0 0 1]
6	[0 1 0 1 1]
7	[0 1 1 1 1]
8	[1 0 0 1 1]
…	…

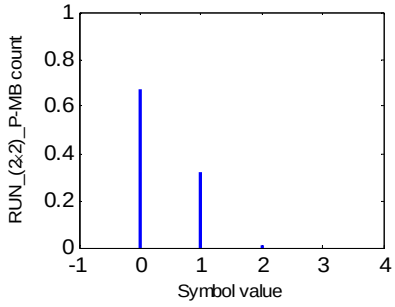
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.5417 bits, (entropy_2.4903 bits)

0	[1]
1	[0 0 0]
2	[0 1 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 1 1 0 0]
6	[0 1 1 1 0]
7	[0 1 1 1 1]
8	[0 1 1 0 1 1]
…	…

(c)

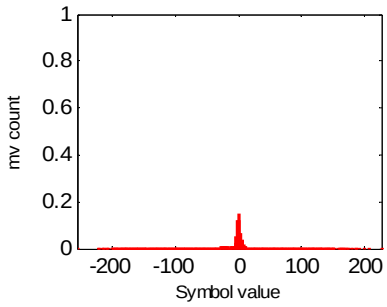


RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.3281 bits, (entropy_0.97319 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

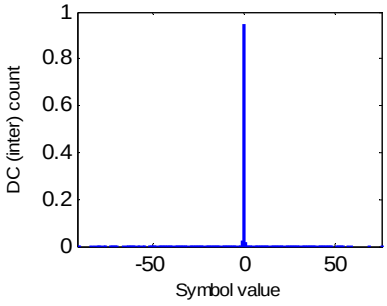
Figure 3.22 Histograms and codebooks (SAD-based) for motion block size 4x4 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



mv codebook ∈ [-256 227] size_379 symbols
avg length_4.8561 bits, (entropy_4.8349 bits)

…	…
-4	[1 0 0 1 1 0]
-3	[1 1 0 0]
-2	[0 0 0 0 1]
-1	[0 1 1]
0	[0 0 1]
1	[1 0 1]
2	[1 1 1 0]
3	[1 0 0 0]
4	[0 0 0 1 0 1]
…	…

(a)

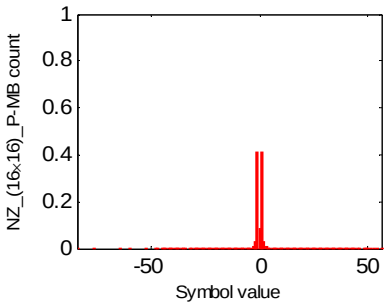


DC codebook ∈ [-92 76] size_136 symbols
avg length_1.1004 bits, (entropy_0.38315 bits)

…	…
-4	[1 0 1 1 1 1 1]
-3	[1 0 1 1 0 1]
-2	[1 0 1 0 0]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 0 1]
3	[1 0 1 1 1 0]
4	[1 0 1 1 0 0 0 0]
…	…

(b)

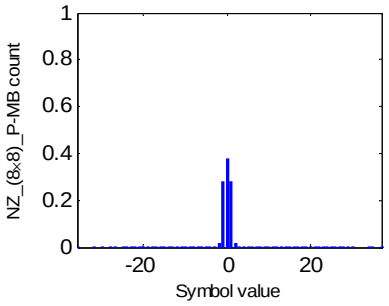
Figure 3.23 Histograms and the corresponding Huffman codebooks for motion block size 16×16 (with SSIM-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-84 56] size_105
avg length_1.9824 bits, (entropy_1.8811 bits)

…	…
-4	[0 1 0 0 1 1 0 1]
-3	[0 1 0 0 1 0 0]
-2	[0 1 0 1]
-1	[1]
0	[0 1 1] EndOfNZ
1	[0 0]
2	[0 1 0 0 0]
3	[0 1 0 0 1 0 1]
4	[0 1 0 0 1 1 1 0]
…	…

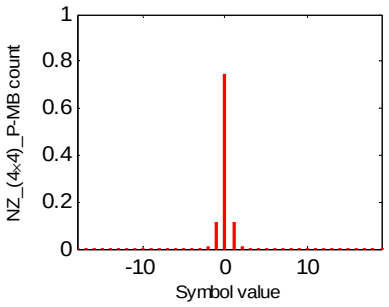
(a)



NZ_(8×8)_P-MB codebook ∈ [-36 37] size_64
avg length_2.1087 bits, (entropy_1.9458 bits)

…	…
-4	[0 0 1 0 1 1 0 1]
-3	[0 0 1 0 1 0 0]
-2	[0 0 1 1]
-1	[0 1]
0	[1] EndOfNZ
1	[0 0 0]
2	[0 0 1 0 0]
3	[0 0 1 0 1 0 1]
4	[0 0 1 0 1 1 1 0]
…	…

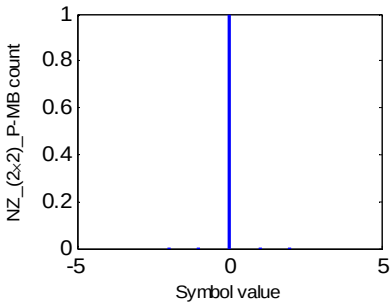
(b)



NZ_(4×4)_P-MB codebook ∈ [-18 19] size_38
avg length_1.4512 bits, (entropy_1.2182 bits)

…	…
-4	[1 0 1 0 1 1 0 0]
-3	[1 0 1 0 1 0 0]
-2	[1 0 1 1]
-1	[1 0 0]
0	[0] EndOfNZ
1	[1 1]
2	[1 0 1 0 0]
3	[1 0 1 0 1 0 1]
4	[1 0 1 0 1 1 1 0]
…	…

(c)

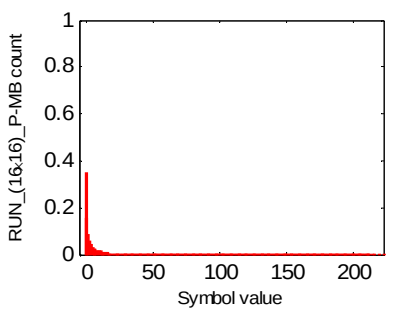


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0012 bits, (entropy_0.010122 bits)

-2	[1 1 1 0]
-1	[1 0]
0	[0] EndOfNZ
1	[1 1 0]
2	[1 1 1 1]

(d)

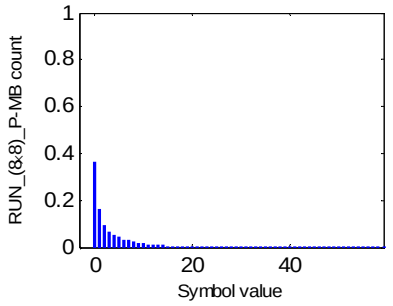
Figure 3.24 Histograms and codebooks (SSIM-based) for motion block size 16×16 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 222] size_217
avg length_3.8171 bits, (entropy_3.7512 bits)

0	[0 0]
1	[0 1 0]
2	[1 1 0]
3	[1 0 0 0]
4	[1 0 1 1]
5	[0 1 1 0 1]
6	[1 0 0 1 0]
7	[1 0 1 0 1]
8	[1 1 1 0 1]
…	…

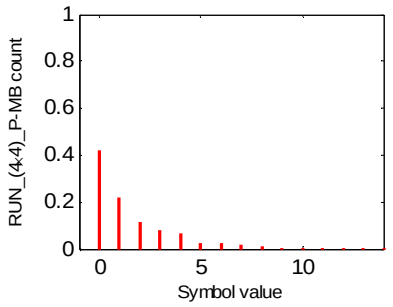
(a)



RUN_(8×8)_P-MB codebook ∈ [0 59] size_60
avg length_3.405 bits, (entropy_3.3353 bits)

0	[1]
1	[0 0 1]
2	[0 0 0 1]
3	[0 1 1 0]
4	[0 0 0 0 0]
5	[0 1 0 0 0]
6	[0 1 0 1 1]
7	[0 1 1 1 1]
8	[0 0 0 0 1 0]
…	…

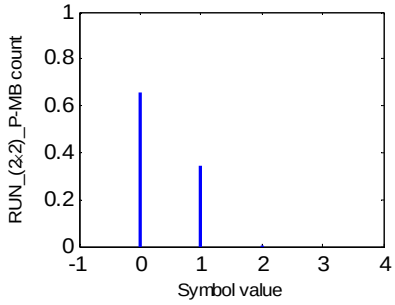
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.5353 bits, (entropy_2.4781 bits)

0	[1]
1	[0 1]
2	[0 0 0 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 0 0 1 0 0]
6	[0 0 0 1 0 1]
7	[0 0 0 1 1 1]
8	[0 0 0 1 1 0 1]
…	…

(c)

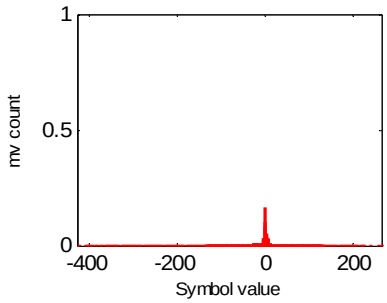


RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.3468 bits, (entropy_0.96992 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

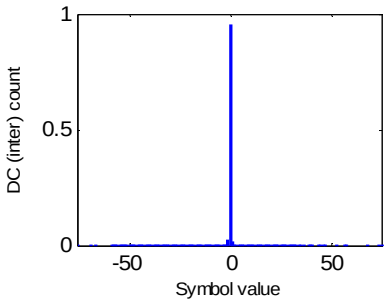
Figure 3.25 Histograms and codebooks (SSIM-based) for motion block size 16×16 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



mv codebook ∈ [-428 264] size_361 symbols
avg length_5.3311 bits, (entropy_5.2956 bits)

…	…
-4	[0 1 1 1 0]
-3	[0 1 0 1 1]
-2	[0 1 1 0 1]
-1	[1 0 1]
0	[0 0 1]
1	[0 0 0 0]
2	[0 1 0 1 0]
3	[1 1 0 1]
4	[0 1 0 0 0]
…	…

(a)

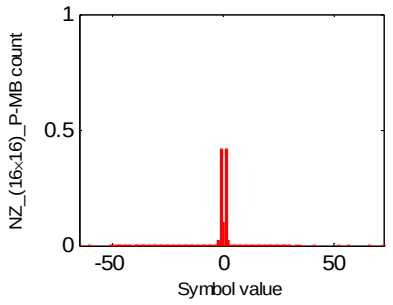


DC codebook ∈ [-76 75] size_112 symbols
avg length_1.0931 bits, (entropy_0.36531 bits)

…	…
-4	[1 0 1 1 1 1 1]
-3	[1 0 1 1 0 1]
-2	[1 0 1 0 0]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 0 1]
3	[1 0 1 1 1 0]
4	[1 0 1 1 1 1 0]
…	…

(b)

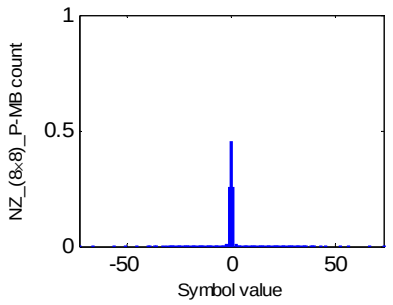
Figure 3.26 Histograms and the corresponding Huffman codebooks for motion block size 8×8 (with SSIM-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-65 72] size_90
avg length_1.8931 bits, (entropy_1.7778 bits)

…	…
-4	[0 1 1 0 1 1 0 1]
-3	[0 1 1 0 1 0 0]
-2	[0 1 1 1]
-1	[0 0]
0	[0 1 0] EndOfNZ
1	[1]
2	[0 1 1 0 0]
3	[0 1 1 0 1 0 1]
4	[0 1 1 0 1 1 1 0]
…	…

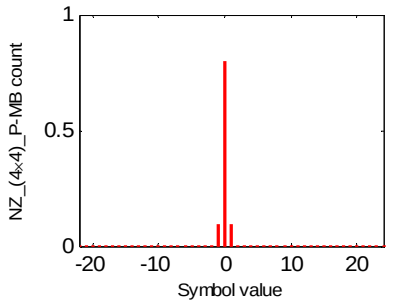
(a)



NZ_(8×8)_P-MB codebook ∈ [-73 73] size_87
avg length_1.9243 bits, (entropy_1.7825 bits)

…	…
-4	[0 0 1 0 1 1 0 1]
-3	[0 0 1 0 1 0 0]
-2	[0 0 1 1]
-1	[0 0 0]
0	[1] EndOfNZ
1	[0 1]
2	[0 0 1 0 0]
3	[0 0 1 0 1 0 1]
4	[0 0 1 0 1 1 1 0]
…	…

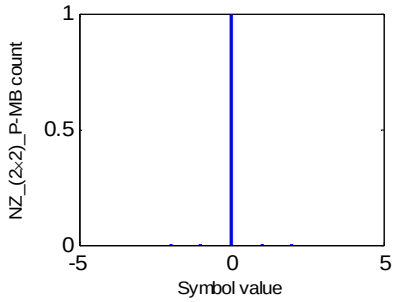
(b)



NZ_(4×4)_P-MB codebook ∈ [-22 24] size_47
avg length_1.3387 bits, (entropy_1.0114 bits)

…	…
-4	[1 0 1 0 1 1 1 0]
-3	[1 0 1 0 1 0 1]
-2	[1 0 1 0 0]
-1	[1 0 0]
0	[0] EndOfNZ
1	[1 1]
2	[1 0 1 1]
3	[1 0 1 0 1 0 0]
4	[1 0 1 0 1 1 0 1]
…	…

(c)

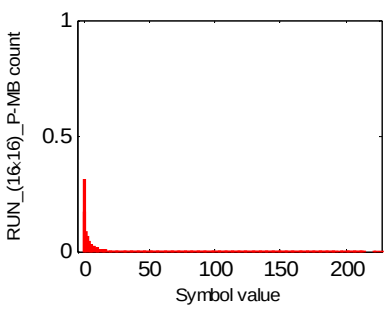


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0012 bits, (entropy_0.010575 bits)

-2	[1 1 1 1]
-1	[1 1 0]
0	[0] EndOfNZ
1	[1 0]
2	[1 1 1 0]

(d)

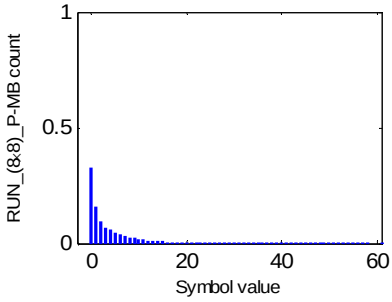
Figure 3.27 Histograms and codebooks (SSIM-based) for motion block size 8x8 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 227] size_218
avg length_3.9279 bits, (entropy_3.8797 bits)

0	[0 0]
1	[0 1 0]
2	[1 1 0]
3	[0 1 1 1]
4	[1 0 1 1]
5	[0 1 1 0 1]
6	[1 0 0 0 1]
7	[1 0 1 0 1]
8	[1 1 1 0 1]
…	…

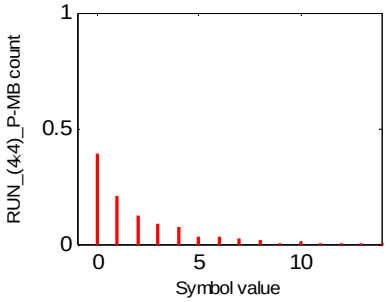
(a)



RUN_(8×8)_P-MB codebook ∈ [0 61] size_60
avg length_3.5815 bits, (entropy_3.5312 bits)

0	[0 0]
1	[0 1 0]
2	[1 1 0]
3	[0 1 1 0]
4	[1 0 0 0]
5	[1 0 1 1]
6	[1 1 1 1]
7	[0 1 1 1 1]
8	[1 0 1 0 0]
…	…

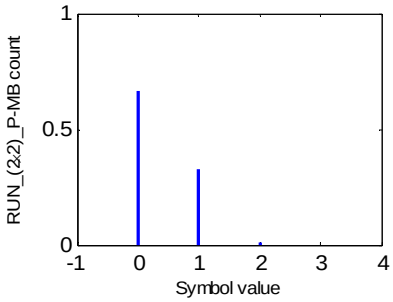
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.6715 bits, (entropy_2.6064 bits)

0	[1]
1	[0 0 0]
2	[0 1 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 1 1 0 1]
6	[0 1 1 1 0]
7	[0 1 1 1 1]
8	[0 1 1 0 0 1]
…	…

(c)

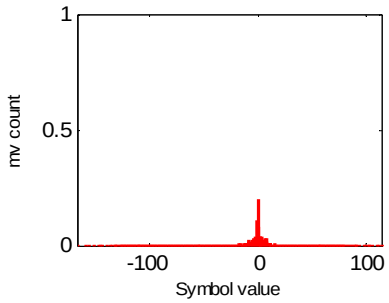


RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.3355 bits, (entropy_0.97007 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

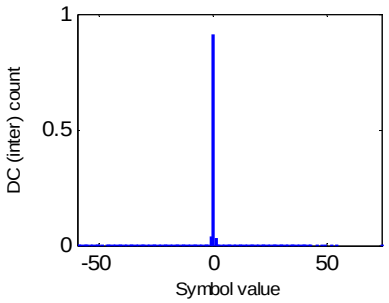
Figure 3.28 Histograms and codebooks (SSIM-based) for motion block size 8×8 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



mv codebook ∈ [-168 115] size_241 symbols
avg length_4.6847 bits, (entropy_4.6558 bits)

…	…
-4	[0 1 0 1 0]
-3	[0 1 1 1 1]
-2	[0 0 1 1 0]
-1	[1 0 0]
0	[1 1]
1	[0 0 0 1]
2	[0 1 0 0 0]
3	[0 0 0 0 1]
4	[0 0 1 0 0]
…	…

(a)

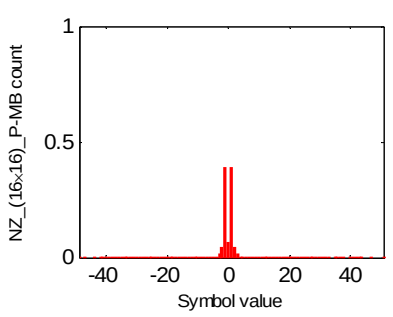


DC codebook ∈ [-60 74] size_109 symbols
avg length_1.2003 bits, (entropy_0.62732 bits)

…	…
-4	[1 0 1 1 0 0 0 0]
-3	[1 0 1 1 1 0]
-2	[1 0 1 0 0]
-1	[1 1]
0	[0]
1	[1 0 0]
2	[1 0 1 0 1]
3	[1 0 1 1 1 1]
4	[1 0 1 1 0 1 1]
…	…

(b)

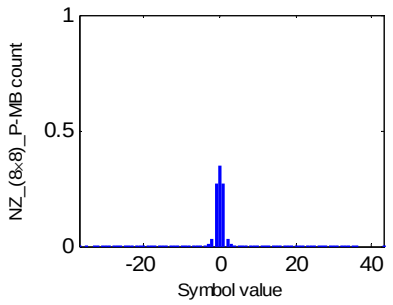
Figure 3.29 Histograms and the corresponding Huffman codebooks for motion block size 4×4 (with SSIM-based training) of, (a) Motion vector, (b) dc coefficient



NZ_(16×16)_P-MB codebk ∈ [-49 51] size_90
avg length_2.2161 bits, (entropy_2.1413 bits)

…	…
-4	[0 1 0 1 0 0 1]
-3	[0 1 0 1 0 1]
-2	[0 1 1 1]
-1	[0 0]
0	[0 1 0 0] EndOfNZ
1	[1]
2	[0 1 1 0]
3	[0 1 0 1 1 0]
4	[0 1 0 1 1 1 0]
…	…

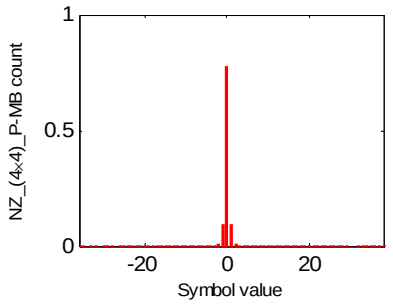
(a)



NZ_(8×8)_P-MB codebook ∈ [-37 43] size_73
avg length_2.3138 bits, (entropy_2.2083 bits)

…	…
-4	[1 1 1 1 1 0]
-3	[1 1 1 0 1]
-2	[1 1 0 0]
-1	[0 1]
0	[0 0] EndOfNZ
1	[1 0]
2	[1 1 0 1]
3	[1 1 1 1 0]
4	[1 1 1 1 1 1]
…	…

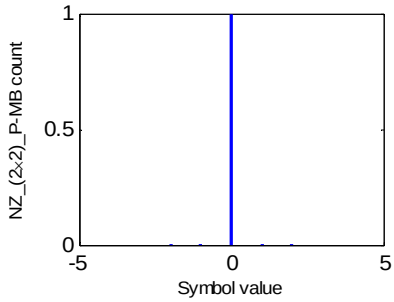
(b)



NZ_(4×4)_P-MB codebook ∈ [-36 38] size_70
avg length_1.4484 bits, (entropy_1.1818 bits)

…	…
-4	[1 0 1 1 0 0 0 0]
-3	[1 0 1 1 1 1]
-2	[1 0 1 0 1]
-1	[1 1]
0	[0] EndOfNZ
1	[1 0 0]
2	[1 0 1 0 0]
3	[1 0 1 1 1 0]
4	[1 0 1 1 0 1 1]
…	…

(c)

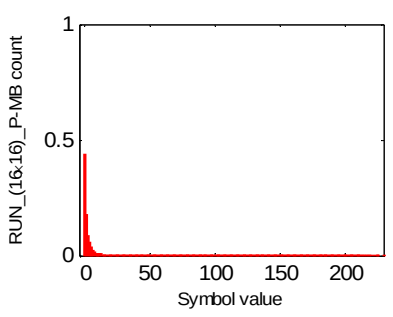


NZ_(2×2)_P-MB codebook ∈ [-2 2] size_5
avg length_1.0031 bits, (entropy_0.023318 bits)

-2	[1 1 1 0]
-1	[1 0]
0	[0] EndOfNZ
1	[1 1 0]
2	[1 1 1 1]

(d)

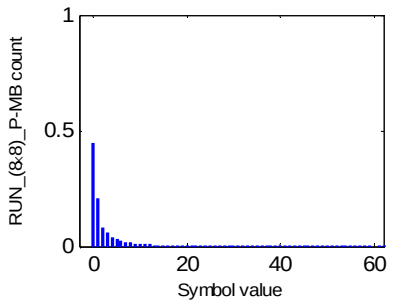
Figure 3.30 Histograms and codebooks (SSIM-based) for motion block size 4×4 of NZ with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2



RUN_(16×16)_P-MB codebk ∈ [0 230] size_224
avg length_3.1196 bits, (entropy_3.0794 bits)

0	[1]
1	[0 0 0]
2	[0 0 1 0]
3	[0 1 0 1]
4	[0 0 1 1 0]
5	[0 1 1 0 0]
6	[0 1 1 1 1]
7	[0 0 1 1 1 1]
8	[0 1 0 0 1 0]
…	…

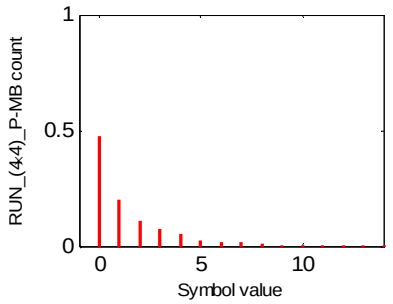
(a)



RUN_(8×8)_P-MB codebook ∈ [0 62] size_62
avg length_2.8557 bits, (entropy_2.8105 bits)

0	[1]
1	[0 1]
2	[0 0 1 0]
3	[0 0 0 0 0]
4	[0 0 0 1 1]
5	[0 0 0 0 1 0]
6	[0 0 0 1 0 0]
7	[0 0 1 1 0 1]
8	[0 0 1 1 1 0]
…	…

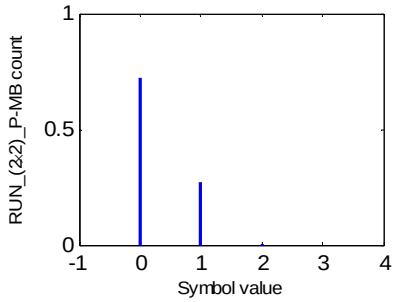
(b)



RUN_(4×4)_P-MB codebook ∈ [0 14] size_15
avg length_2.4066 bits, (entropy_2.365 bits)

0	[1]
1	[0 1]
2	[0 0 0 0]
3	[0 0 1 0]
4	[0 0 1 1]
5	[0 0 0 1 0 0]
6	[0 0 0 1 1 0]
7	[0 0 0 1 1 1]
8	[0 0 0 1 0 1 1]
…	…

(c)



RUN_(2×2)_P-MB codebook ∈ [0 2] size_3
avg length_1.2777 bits, (entropy_0.89812 bits)

0	[0]
1	[1 0]
2	[1 1]

(d)

Figure 3.31 Histograms and codebooks (SSIM-based) for motion block size 4×4 of RUN with transform sizes, (a) 16×16, (b) 8×8, (c) 4×4, (d) 2×2

In this chapter, interframe coding elements of the proposed codec are implemented and Huffman entropy codebooks for a macroblock on P-frame are created. Reference frame interpolations on luma and chroma frames are explained. Method of motion searches in fast motion estimation is explained. To obtain the motion vector in motion prediction, block matching based on SAD and SSIM are explained in detail. Encoded bit stream format for a macroblock on P-frame is introduced with the coefficients for Huffman entropy codebooks. Then, Huffman codebooks with the coefficients and the corresponding binary codes are created from five test sequences and are used for interframe encoding.

CHAPTER 4

SIMULATION RESULTS

4.1 Intraframe Coding

Simulation is performed on four CIF test sequences (Football, Harbour (Demografx), Mobile, and Stefan) [33] (Fig. 4.1). Parameters used in the codec are given in Table 4.1. The rate-distortions (RGB-PSNR versus bit rate, Y-PSNR versus bit rate, Cb-PSNR versus bit rate, and Cr-PSNR versus bit rate) of the codec are compared with the results from JM12.0 [7][8], JPEG2000 [9], and JPEG [10] in Figures 4.2–4.5. Each point in the rate-distortion (RD) plot is obtained by changing the quantization parameter (*QP*) in each simulation. PSNR is obtained from the decoded sequence. Bit rate is calculated from the encoded bits (or encoded file size in bytes) and the frame rate. Tables 4.2–4.3 compare the RD curves using BD-PSNR [18] on luminance and RGB components. In Figure 4.3 and Table 4.2, the proposed codec performs better than H.264 reference software (JM) on Football, Harbour, and Stefan, since BD-PSNR has a positive value. For a Mobile sequence (Fig. 4.3(c)), JM gives better results at high bit rate. In Figure 4.2, PSNR of the true color decoded sequence (RGB-PSNR) is calculated. The proposed codec performance is better than JM for Harbour and Stefan sequences.

Consider the number of operations used in transform of a (16×16) macroblock. H.264-(4×4) integer transform requires 16 adds and 4 shifts [29]. H.264-(8×8) integer transform requires 64 adds and 20 shifts [29]. The simplified order-16 ICT requires 150 adds and 32 shifts [31]. The number of operations in transform process depends on the configuration of a composite block. The proposed encoder requires between 1.59 and 1.83 times the number of operations used in the H.264 baseline. The proposed decoder requires at most 5 percents more

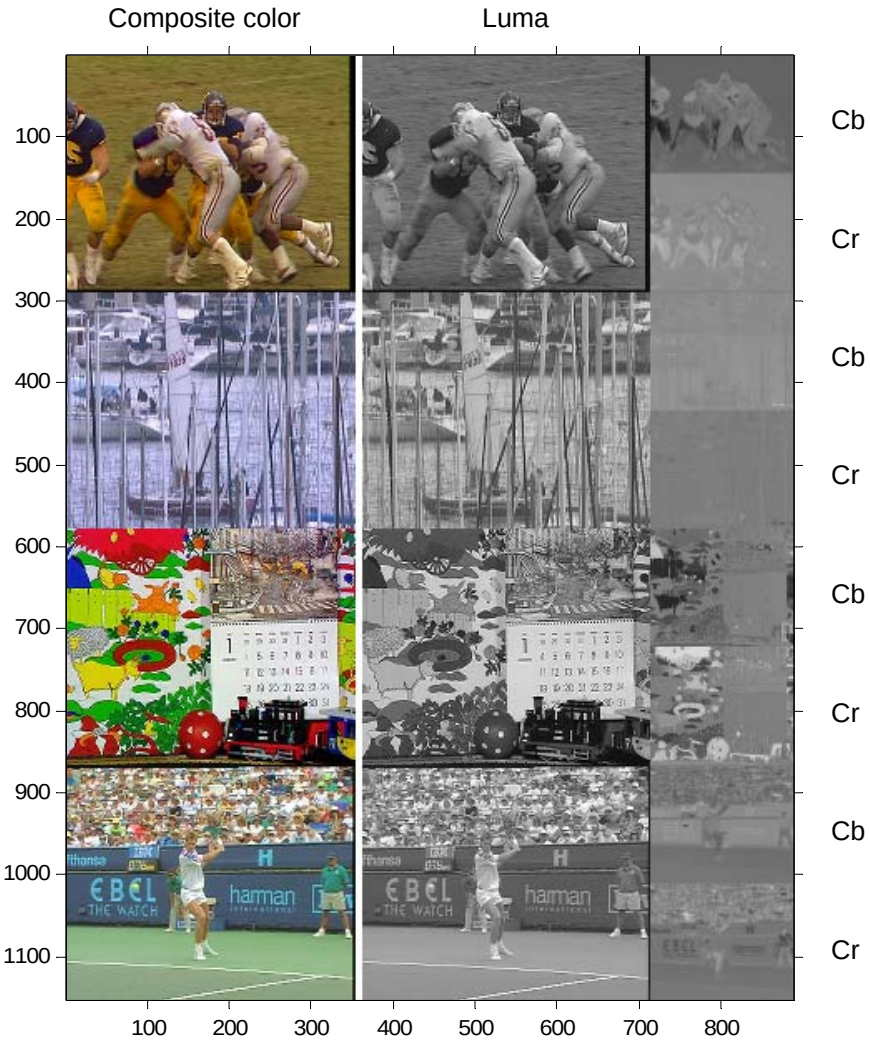


Figure 4.1 Four CIF sequences used in the simulation, from top row, Football frame 50, Harbour frame 0, Mobile frame 80, and Stefan frame 15.

number of operations. At the minimum, the proposed decoder requires 43 percents less compared to H.264 baseline profile. From the example of a composite luminance block in Figure 2.8, at the encoder side, the number of operations increases by 1.82 times. At the decoder side, the number of operations increases by 2.5% compared to H.264 baseline profile.

Table 4.1 Parameter Setting for the Proposed Codec Simulation on Intraframe Coding

Test sequence type (H.264 level 1.3)	CIF (288×352 pixels for luma) at 30 fps, YCbCr 4:2:0 (144×176 pels for chroma)
Frame encoding structure	III··· (9 I-frames)
Macroblock size	(16×16) for luma, (8×8) for chroma
Macroblock partition into composite blocks	Luma sub-block sizes (16×16), (8×8), and (4×4) Chroma sub-block sizes (8×8), (4×4) and (2×2)
Macroblock header (Block size selection bits)	1 or 5 bits depend on partition structure of a macroblock
Spatial prediction (I-frame)	Four spatial modes for luma block (16×16), DC spatial mode for chroma block (8×8)
Distortion measurement (for spatial prediction)	SAD (sum of absolute difference)
Transforms	Sizes (16×16), (8x8), and (4x4) integer transforms for luma sub-block, Sizes (8×8), (4×4), and (2×2) integer transforms for chroma sub-block
Quantization parameters (Uniform quantization)	30–42 (Step sizes are 20–80)
Run-length encoding	(RUN, NZ) with EndOfNZ symbol for a sub-block ending
Entropy encoding	Huffman variable-length code
Simulation tool	MATLAB

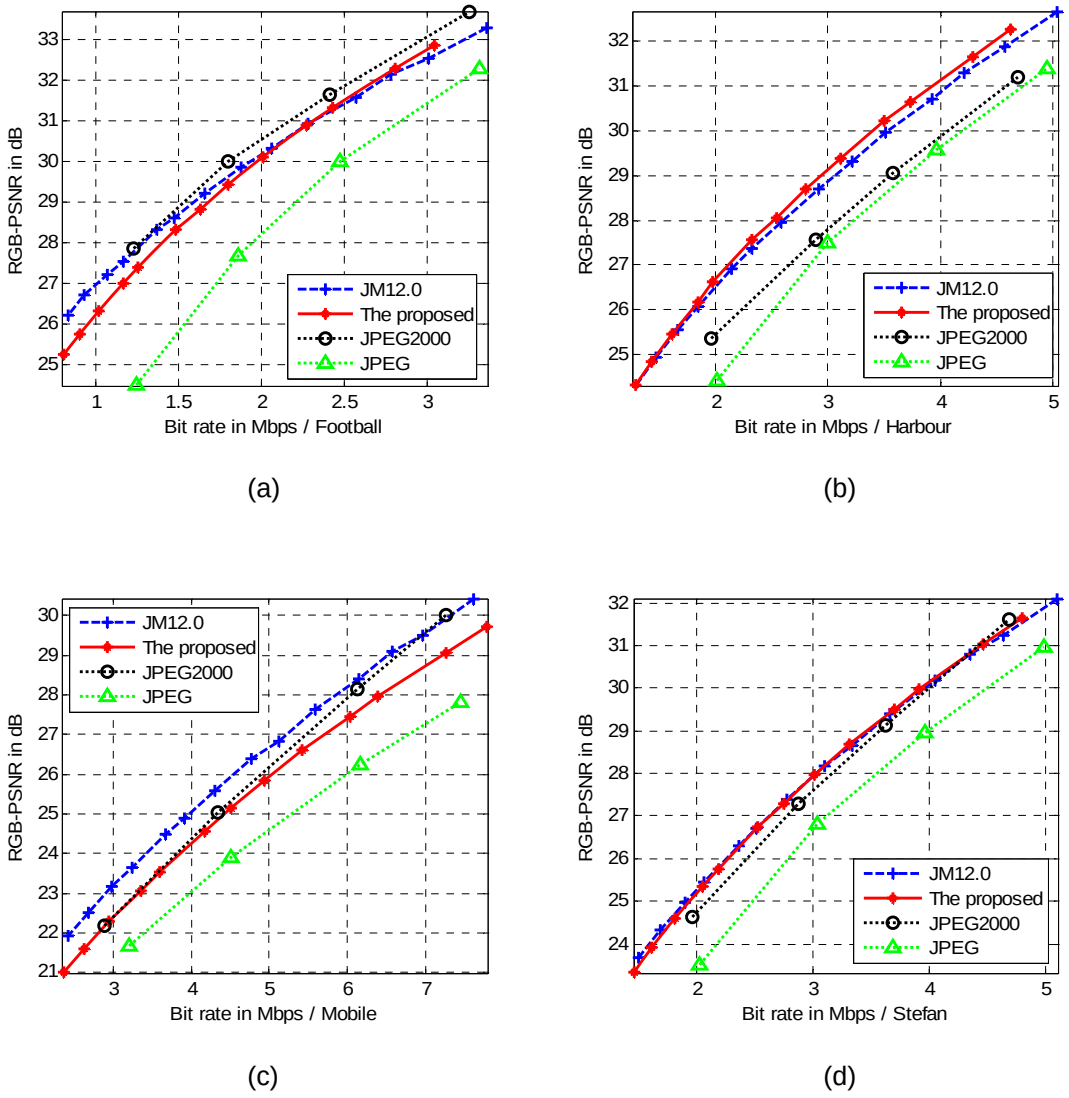


Figure 4.2 RGB-PSNR vs. bit rate comparisons of the proposed codec, JM12.0, JPEG2000, and JPEG, with four test sequences. (a) Football, (b) Harbour, (c) Mobile, and (d) Stefan

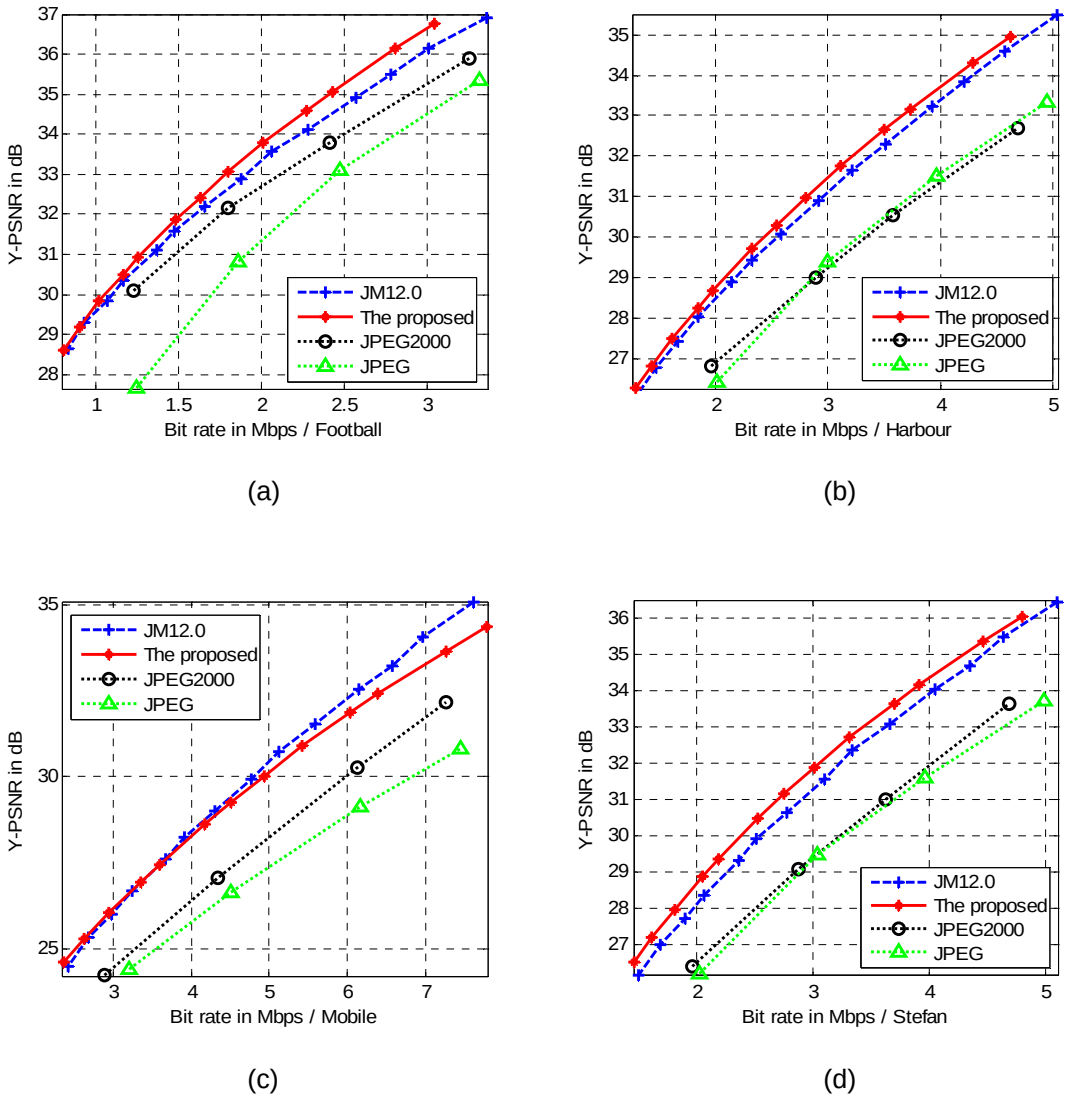


Figure 4.3 Y-PSNR vs. bit rate comparisons of the proposed codec, JM12.0, JPEG2000, and JPEG, with four test sequences. (a) Football, (b) Harbour, (c) Mobile, and (d) Stefan

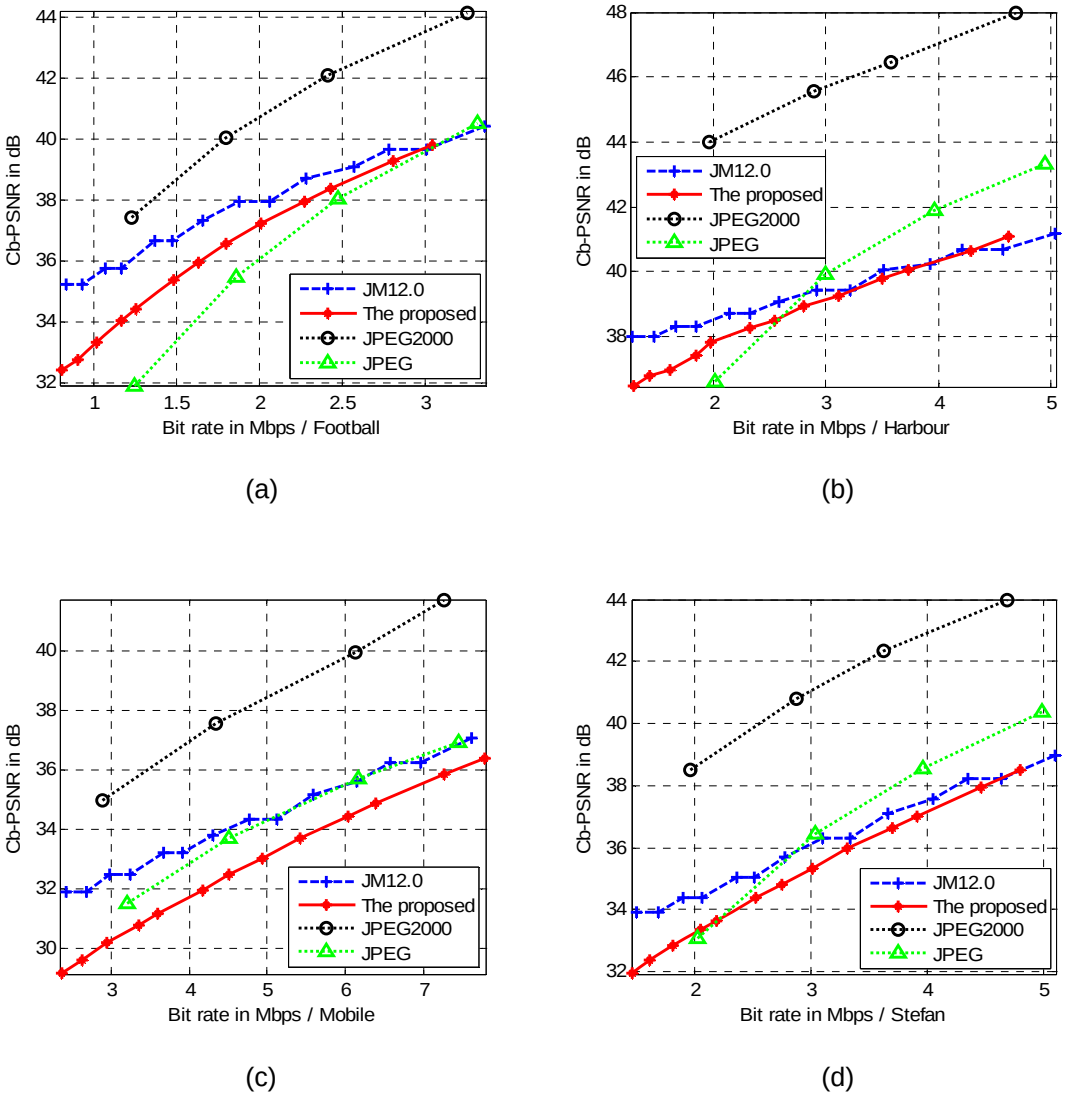


Figure 4.4 Cb-PSNR vs. bit rate comparisons of the proposed codec, JM12.0, JPEG2000, and JPEG, with four test sequences. (a) Football, (b) Harbour, (c) Mobile, and (d) Stefan

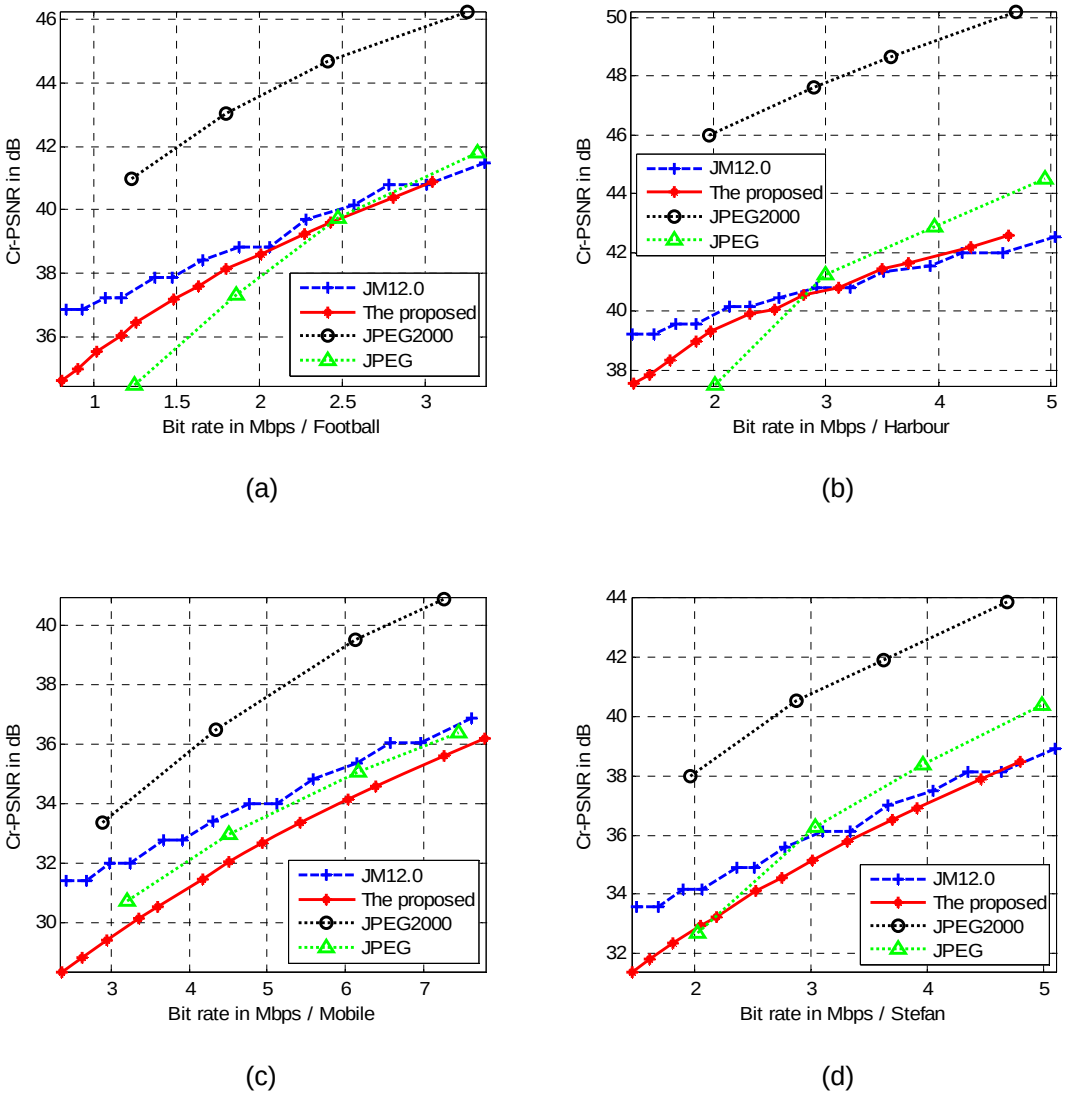


Figure 4.5 Cr-PSNR vs. bit rate comparisons of the proposed codec, JM12.0, JPEG2000, and JPEG, with four test sequences. (a) Football, (b) Harbour, (c) Mobile, and (d) Stefan

Table 4.2 BD-PSNR Comparison on RD Curves of Luminance Component with the Proposed Codec

	Football		Harbour		Mobile		Stefan	
	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)
JM12.0	0.46	-6.79	0.34	-4.37	-0.27	2.72	0.53	-5.95
JPEG2000	1.04	-15.97	2.21	-25.88	1.78	-18.78	2.37	-24.84
JPEG	2.37	-26.34	2.12	-24.04	2.70	-28.36	2.54	-26.47

Table 4.3 BD-PSNR Comparison on RD Curves of RGB Component with the Proposed Codec

	Football		Harbour		Mobile		Stefan	
	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)	BD-PSNR (dB)	BD-Bit rate (%)
JM12.0	-0.04	0.57	0.26	-3.90	-0.80	11.25	0.01	-0.15
JPEG2000	-0.47	8.36	1.29	-16.72	-0.31	3.64	0.30	-3.82
JPEG	1.75	-19.88	1.62	-19.43	1.27	-15.74	1.29	-14.44

4.2 Interframe Coding With SAD and SSIM

Simulations are performed on nine CIF test sequences [33] Football, Harbour, Mobile, Stefan, Akiyo, Ice, Irene, Paris, and Tempete (Figs. 4.1 and 4.6). Parameters used in the codec are given in Table 4.4. For interframe coding, structural similarity (SSIM) is applied inside motion estimation to obtain distortion cost. As the rate-distortion plot, SSIM is also used to obtain the quality measurement of the reconstructed (decoded) video frames. Rate-distortions are compared in terms of RGB-PSNR versus bit rate, and RGB-SSIM versus bit rate. Four combinations with two types of trained codebooks (SAD-based and SSIM-based) and with two types of block-matching distortion (SAD-based and SSIM-based) are simulated in the proposed interframe coding. Each simulation point of a rate-distortion (RD) plot is obtained by changing the quantization parameter (*QP*). PSNR and SSIM are calculated from the decoded sequence. The SSIM parameter setting is given in Table 4.5. Bit rate (bits per second) is calculated from the encoded bits and the given frame rate at 30 frames per second. The results are shown in Figures 4.7–4.12 on RGB components for motion block sizes (16×16), (8×8), and (4×4). Figures 4.13–4.30 show the results on Y, Cb, and Cr components.

Figure 4.6 Five CIF sequences for interframe encoding, from top row, Akiyo frame 30, Ice frame 200, Irene frame 130, Paris frame 70, and Tempete frame 100. From left column, show true color, luma, Cb (top), and Cr (bottom) components

Table 4.4 Parameter Setting for the Proposed Codec Simulation on IPP··· Encoding

Test sequence type	CIF (288×352 pels for luma) at 30 fps, YCbCr 4:2:0 (144×176 pels for chroma)
Frame encoding structure	IPP··· (9 frames)
I-frame encoding	Same parameter setting in Table 4.2
Macroblock size	(16×16) for luma, (8×8) for chroma
Motion estimation (P-frame)	Block matching motion estimation on fixed block sizes (16×16), (8×8), and (4×4) of luma
Number of reference frame	1 frame (previously decoded frame)
Motion vector resolution and reference frame interpolation	1/4 pixels for luma, 1/8 pixels for chroma
Motion search range	–32 to +32 pixels
Motion search	UMHexagonS for integer-pel resolution, CBFPS for fractional-pel resolution
Distortion measurement	SAD and (1 – SSIM)
SSIM calculation parameter	$C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$, $C_3 = C_2 / 2$, $K_1 = 0.01$, $K_2 = 0.03$, $L = 255$, $\alpha = \beta = \gamma = 1$ Local window size: same as block size (16×16), (8x8), and (4×4). Without local window move.
Transform	Sizes (16×16), (8x8), and (4x4) integer transforms for luma sub-block, Sizes (8×8), (4×4), and (2×2) integer transforms for chroma sub-block
Quantization parameter (uniform quantization)	30–42 (Step sizes are 20–80)
Run-length encoding	(RUN, NZ) with EndOfNZ symbol for a sub-block ending
Entropy encoding	Huffman variable-length code
Simulation tool	MATLAB

Table 4.5 Parameter Setting for Reconstructed Video Quality Measurement Using SSIM

Local window size	(11×11)
Window moving	Local window moves pixel by pixel
SSIM parameter (for each block)	$C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$, $C_3 = C_2 / 2$, $K_1 = 0.01$, $K_2 = 0.03$, $L = 255$, $\alpha = \beta = \gamma = 1$
Measuring component (for entire video frames)	For Y, Cb, and Cr components, separate measurement with average SSIM. For R, G, and B components, combined measurement with equal weight.

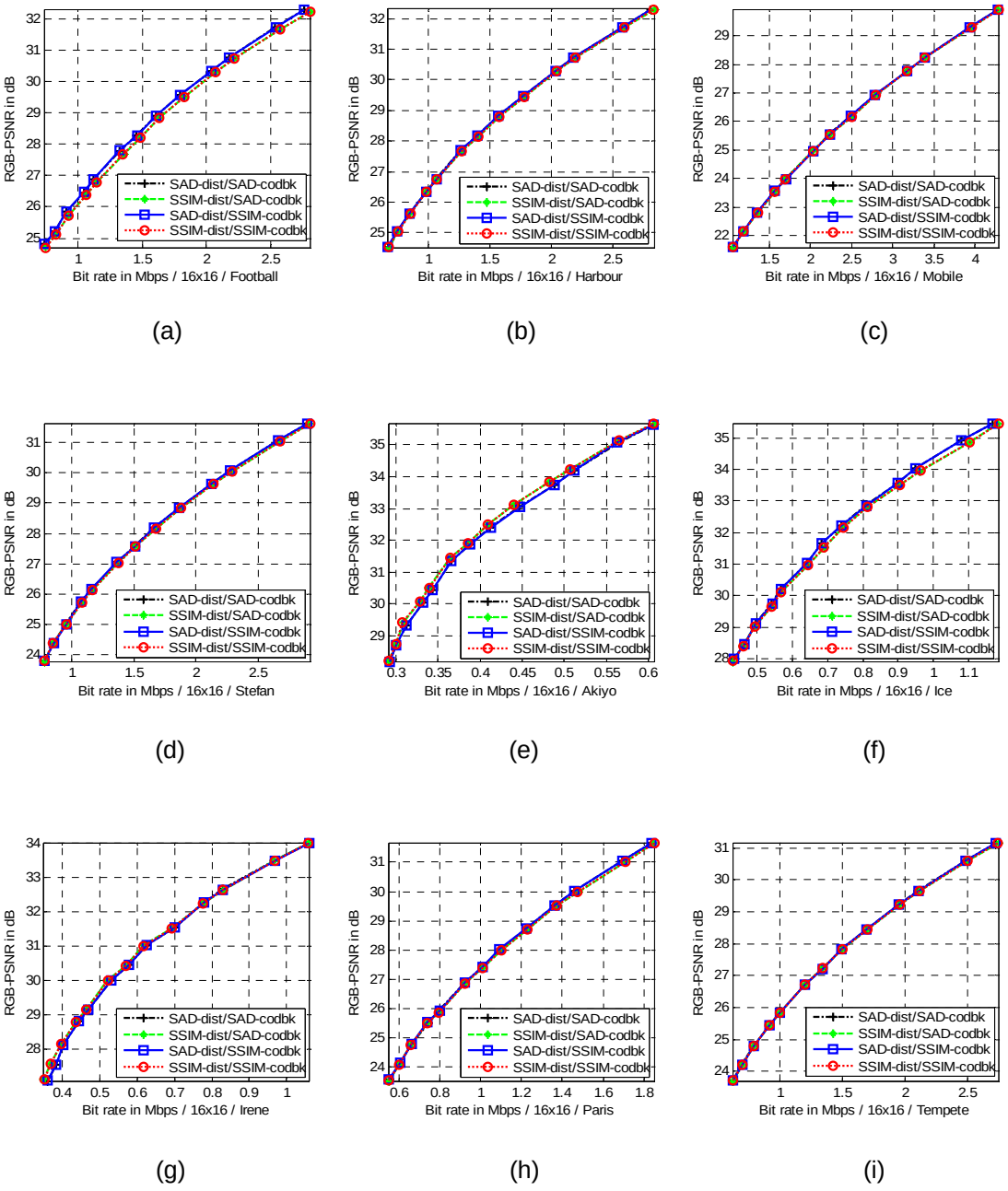
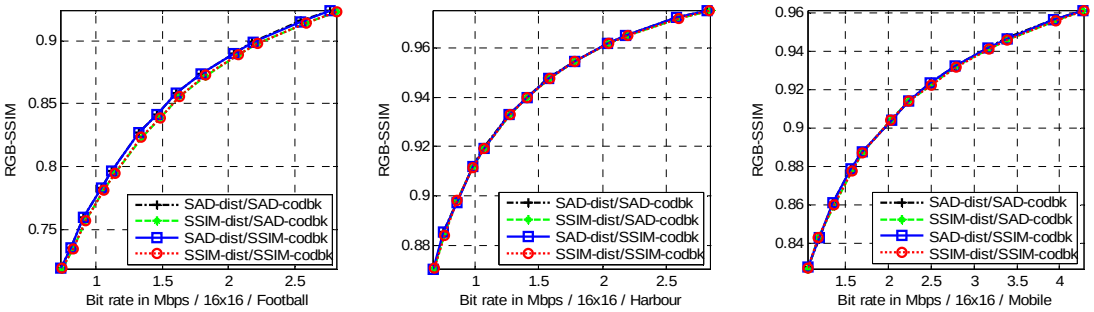


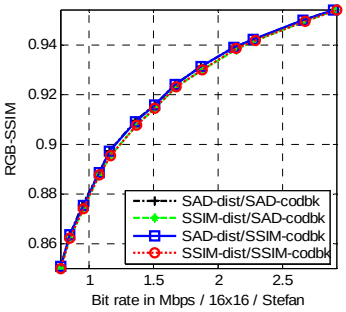
Figure 4.7 RGB-PSNR vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete



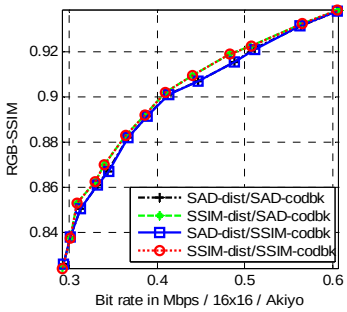
(a)

(b)

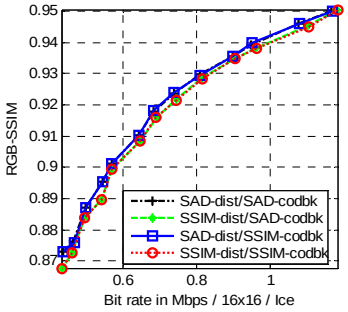
(c)



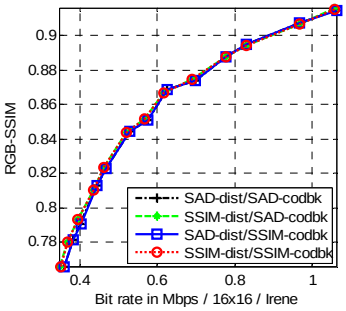
(d)



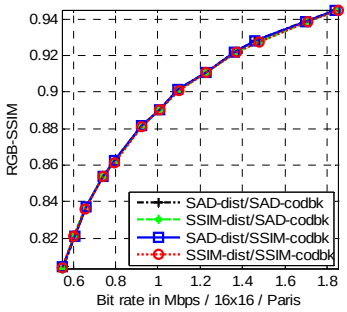
(e)



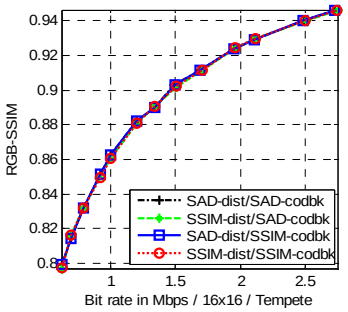
(f)



(g)



(h)



(i)

Figure 4.8 RGB-SSIM vs. bit rate comparison, with motion block size (16×16) of 9 sequences.
(a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice,
(g) Irene, (h) Paris, and (i) Tempete

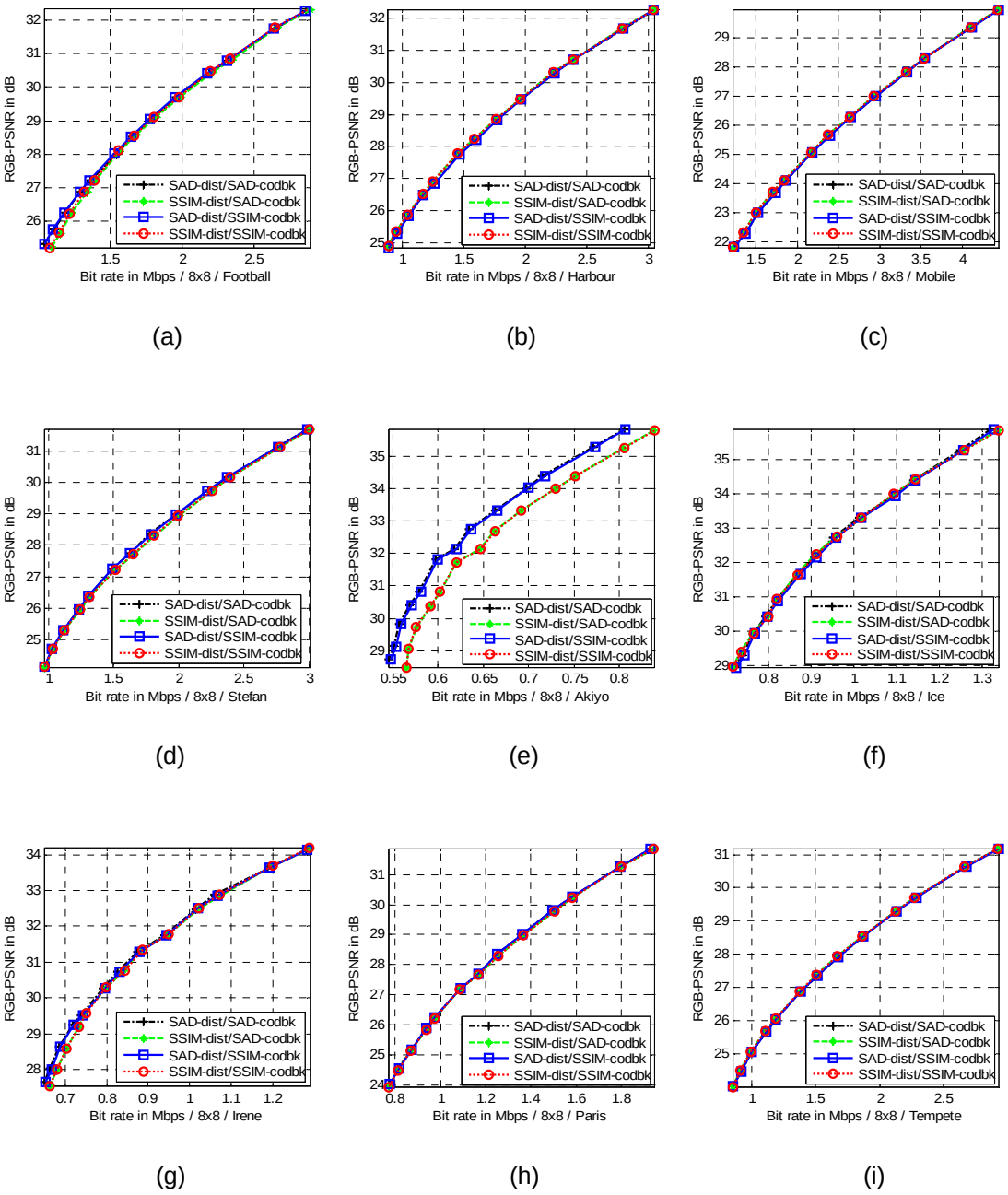


Figure 4.9 RGB-PSNR vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

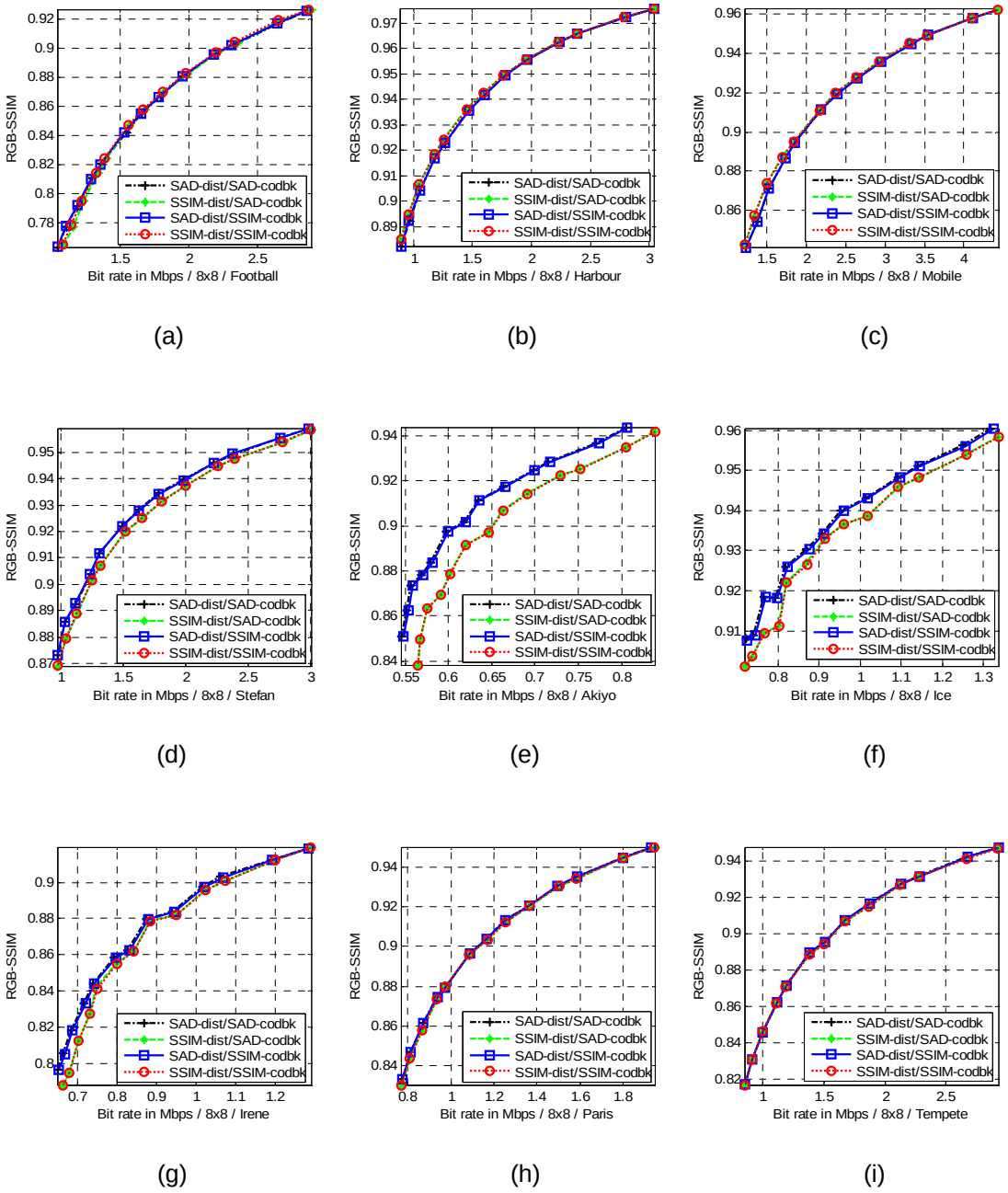


Figure 4.10 RGB-SSIM vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

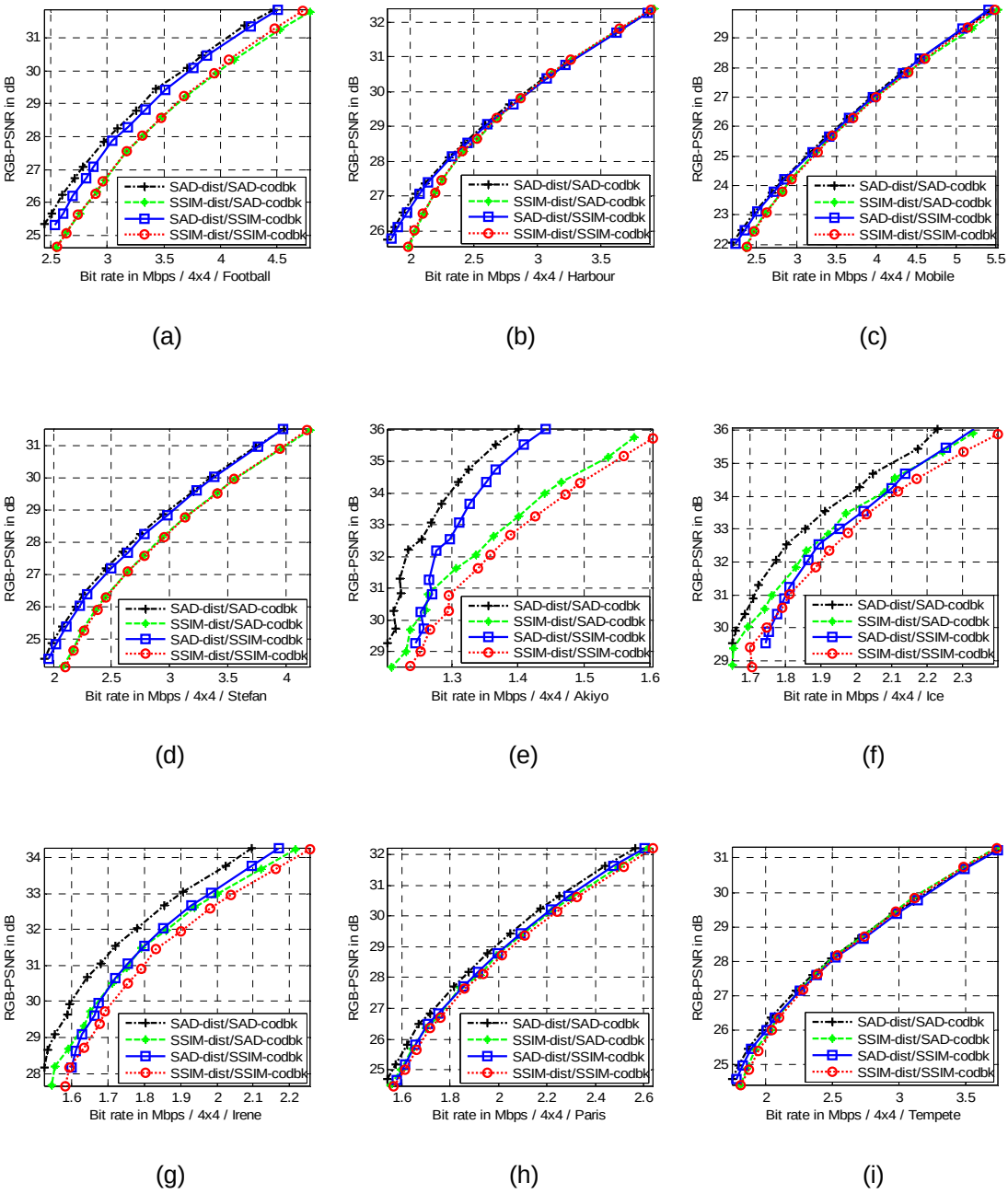


Figure 4.11 RGB-PSNR vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

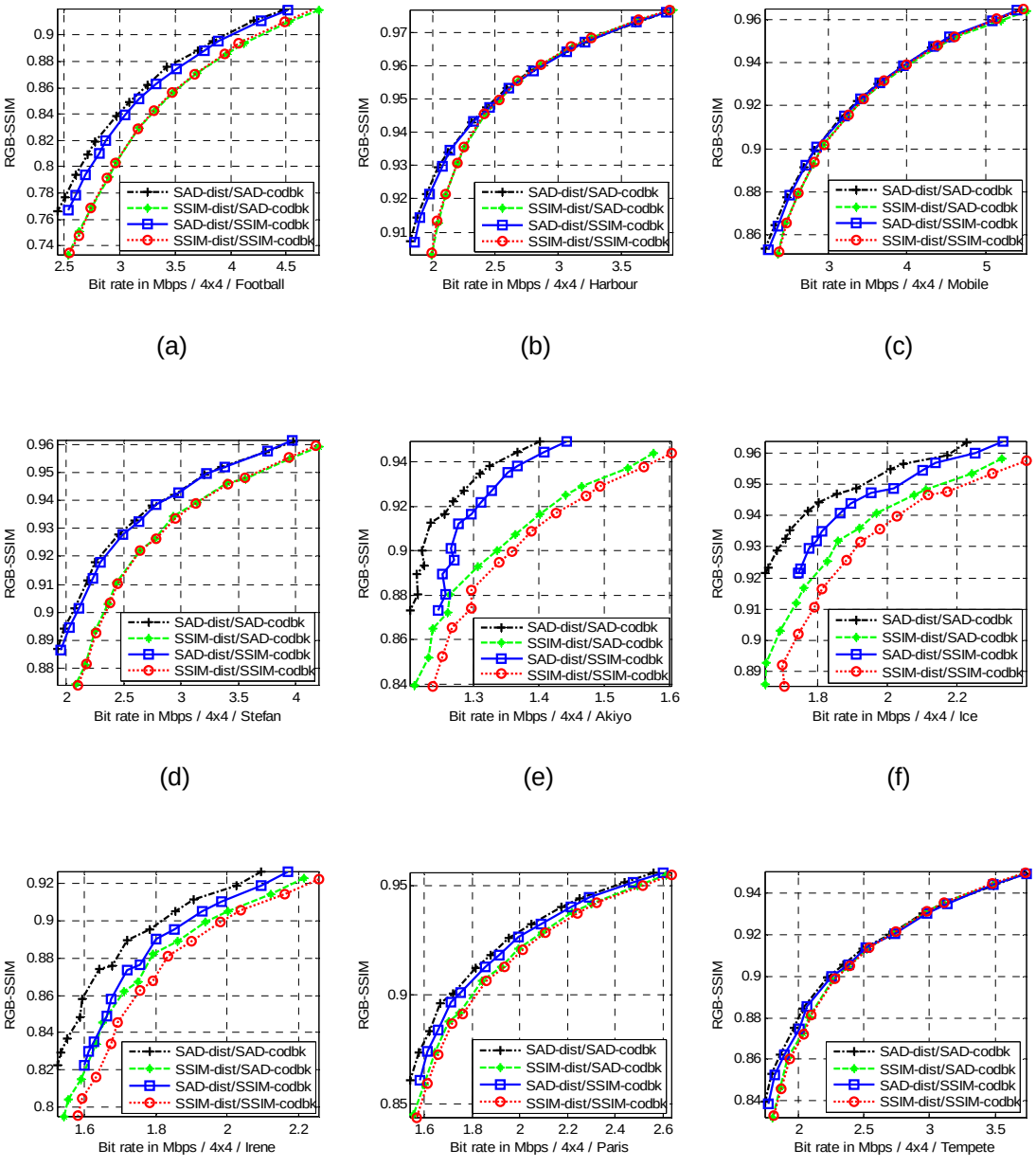


Figure 4.12 RGB-SSIM vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

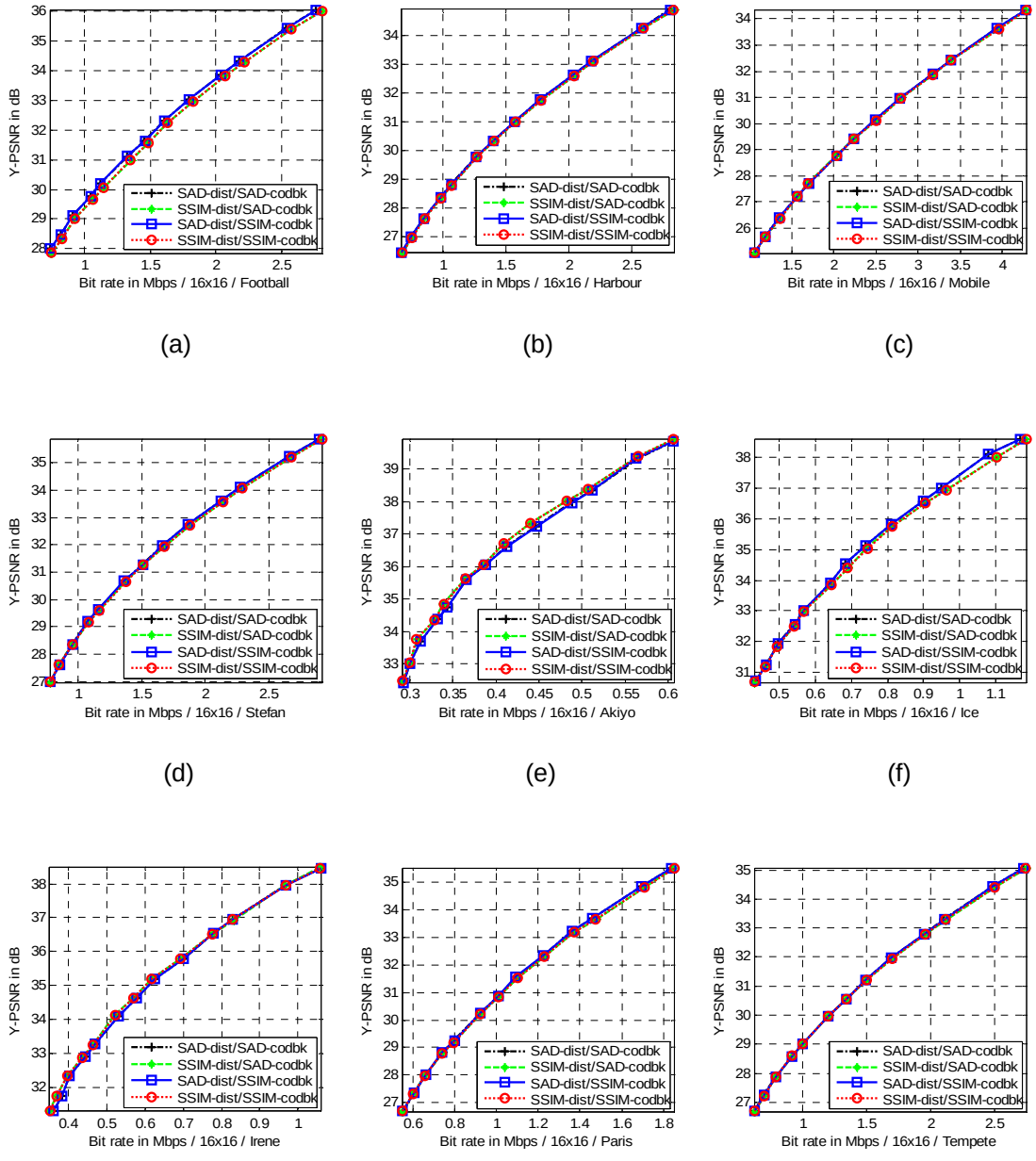


Figure 4.13 Y-PSNR vs. bit rate comparison, motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

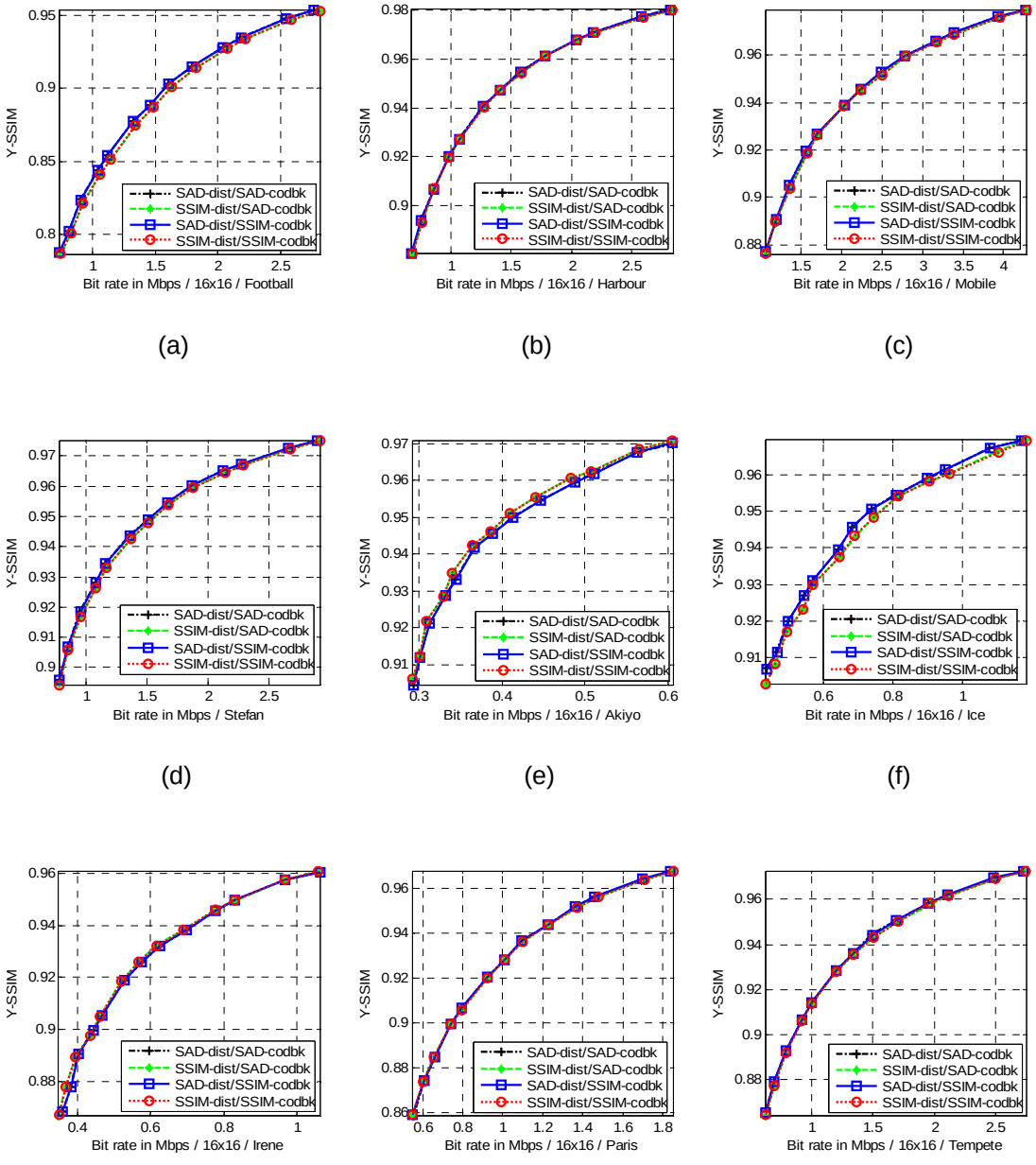


Figure 4.14 Y-SSIM vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

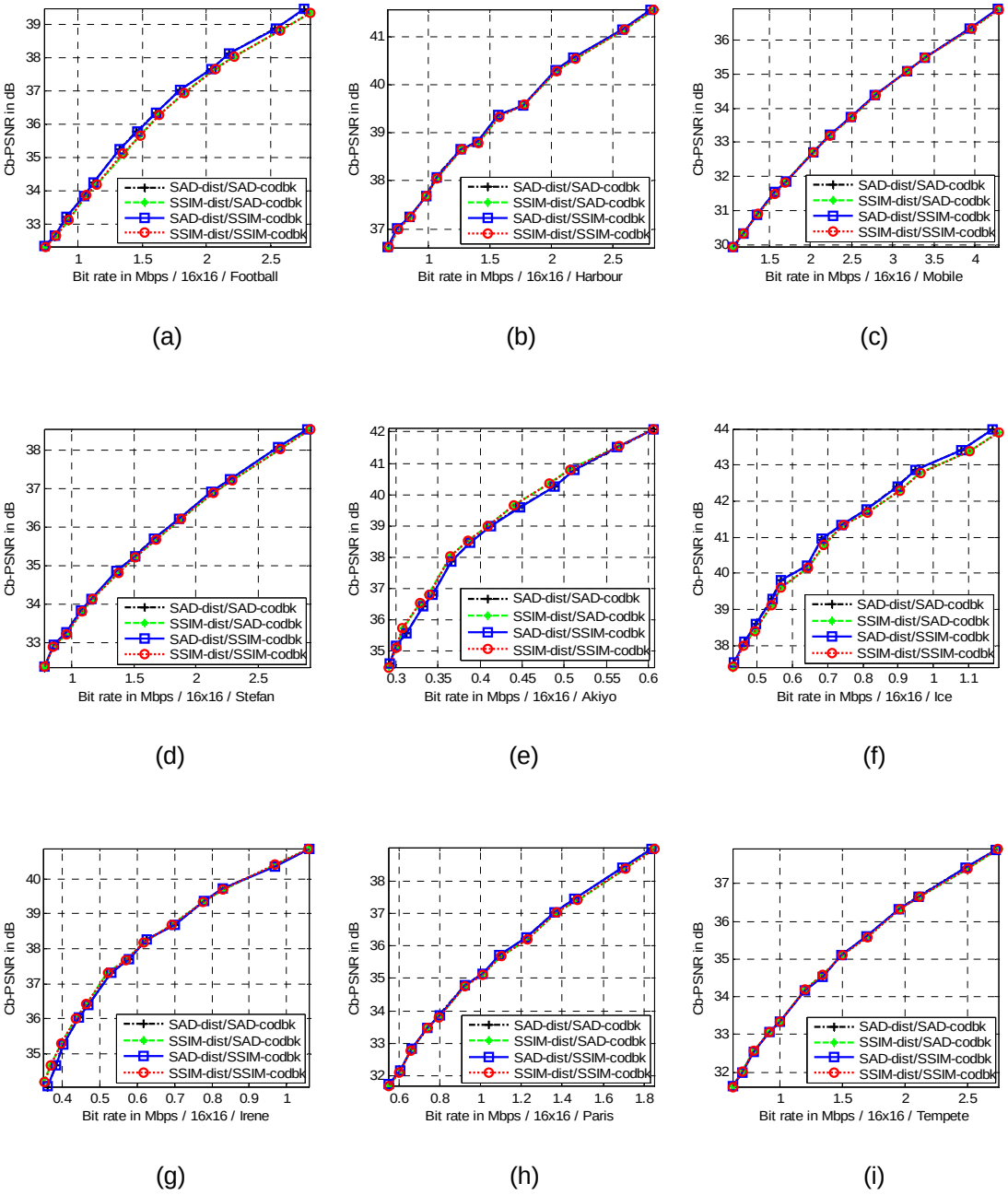


Figure 4.15 Cb-PSNR vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

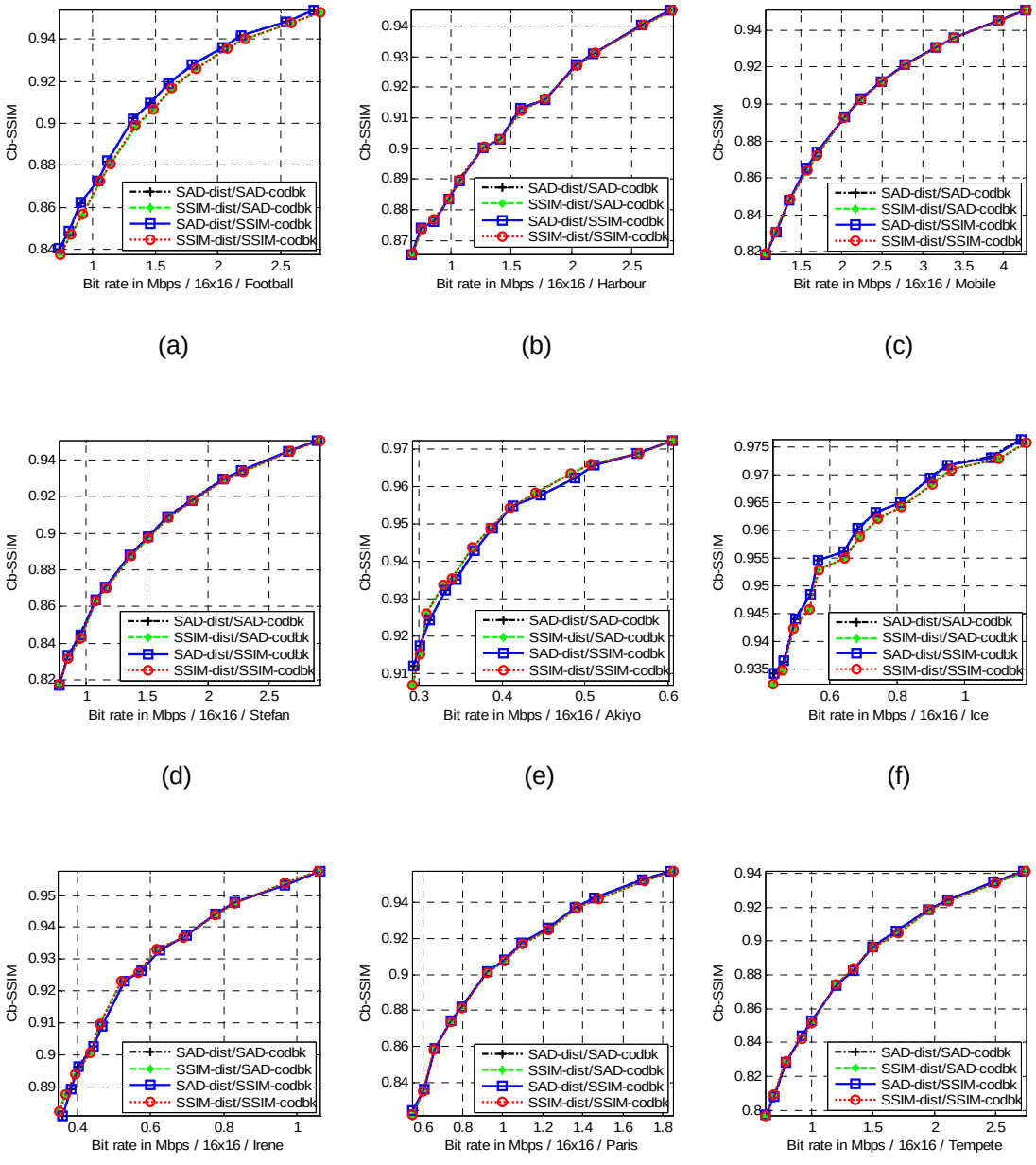


Figure 4.16 Cb-SSIM vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

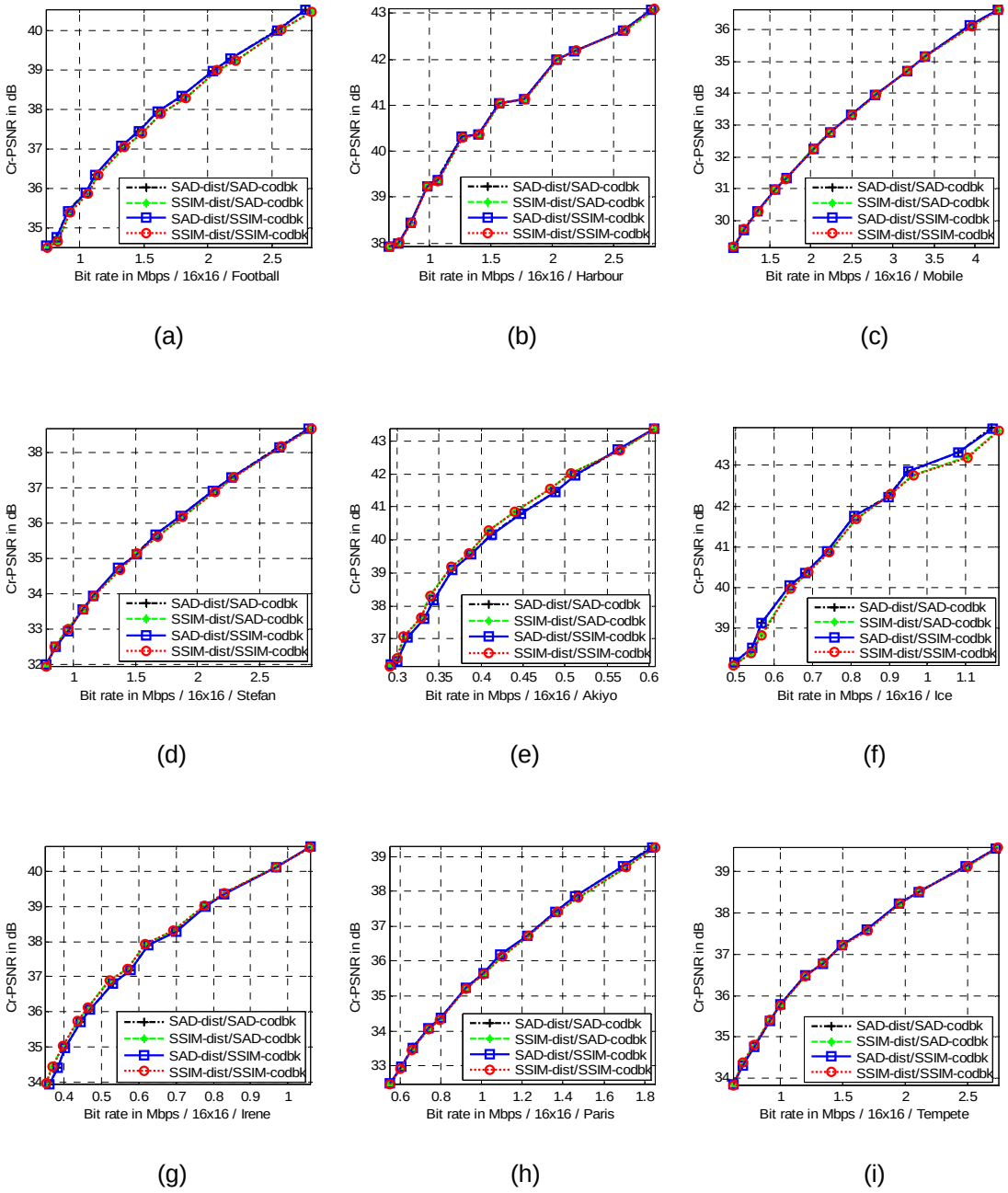


Figure 4.17 Cr-PSNR vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

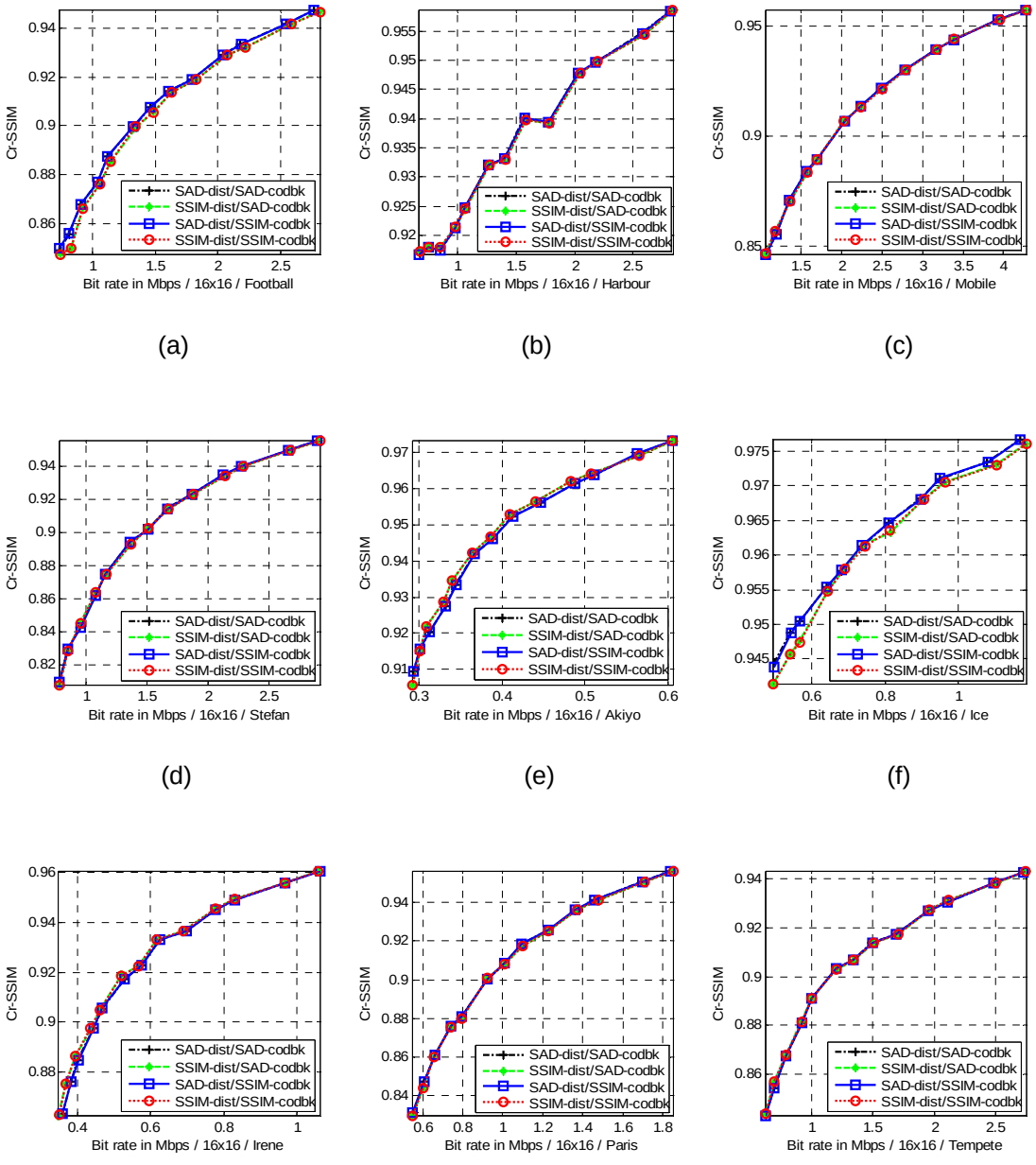


Figure 4.18 Cr-SSIM vs. bit rate comparison, with motion block size (16×16) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

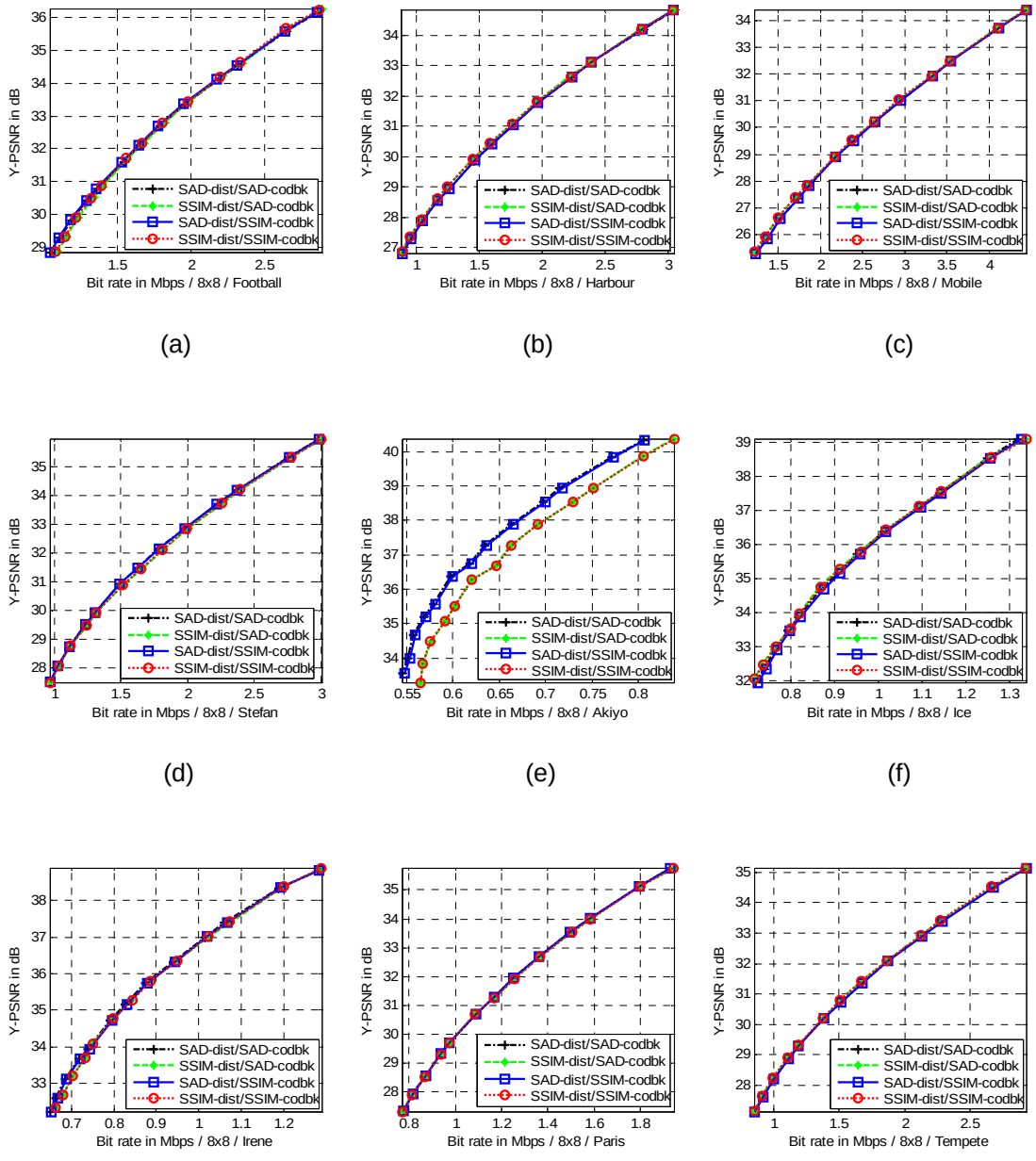


Figure 4.19 Y-PSNR vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

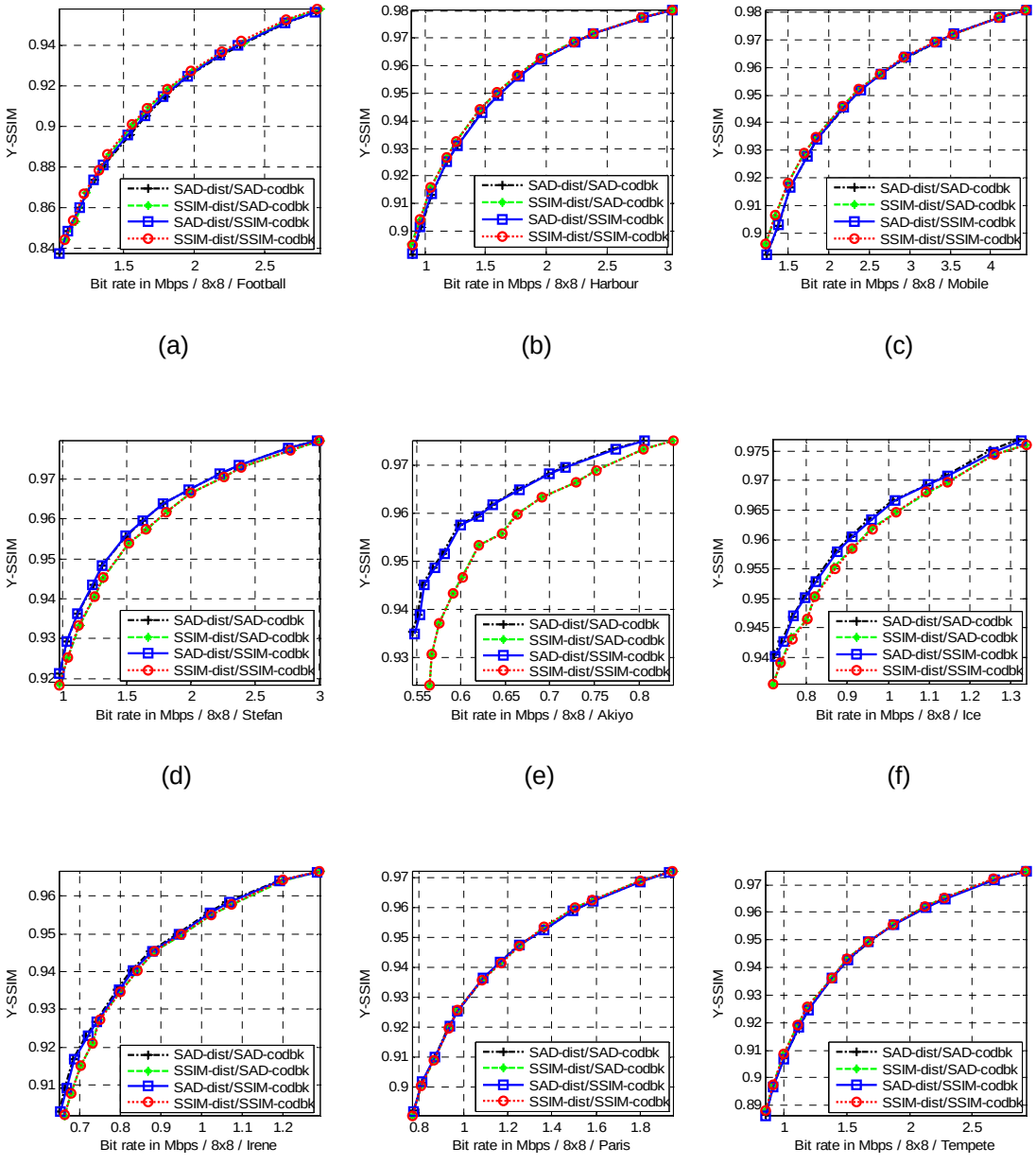


Figure 4.20 Y-SSIM vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

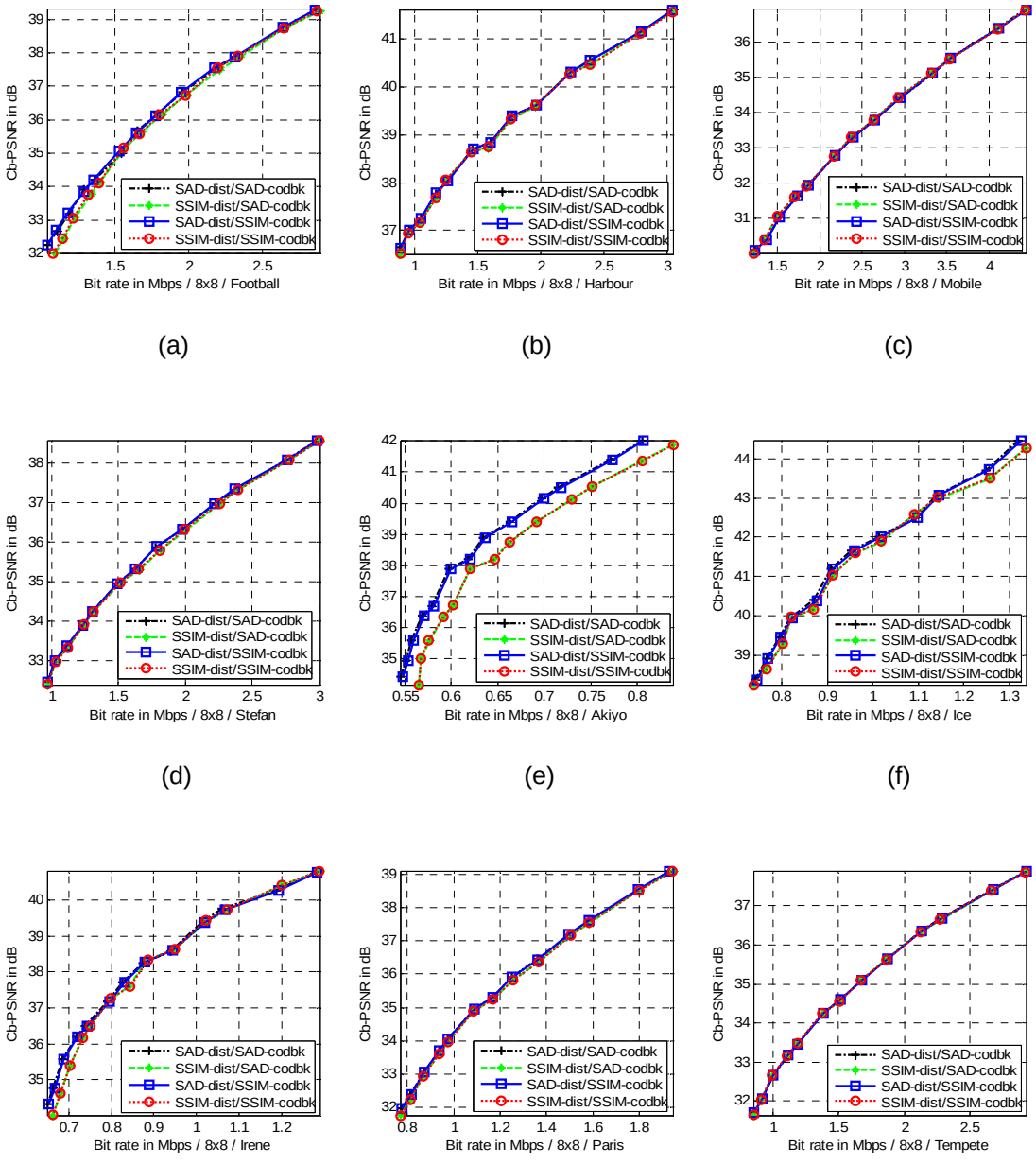


Figure 4.21 Cb-PSNR vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

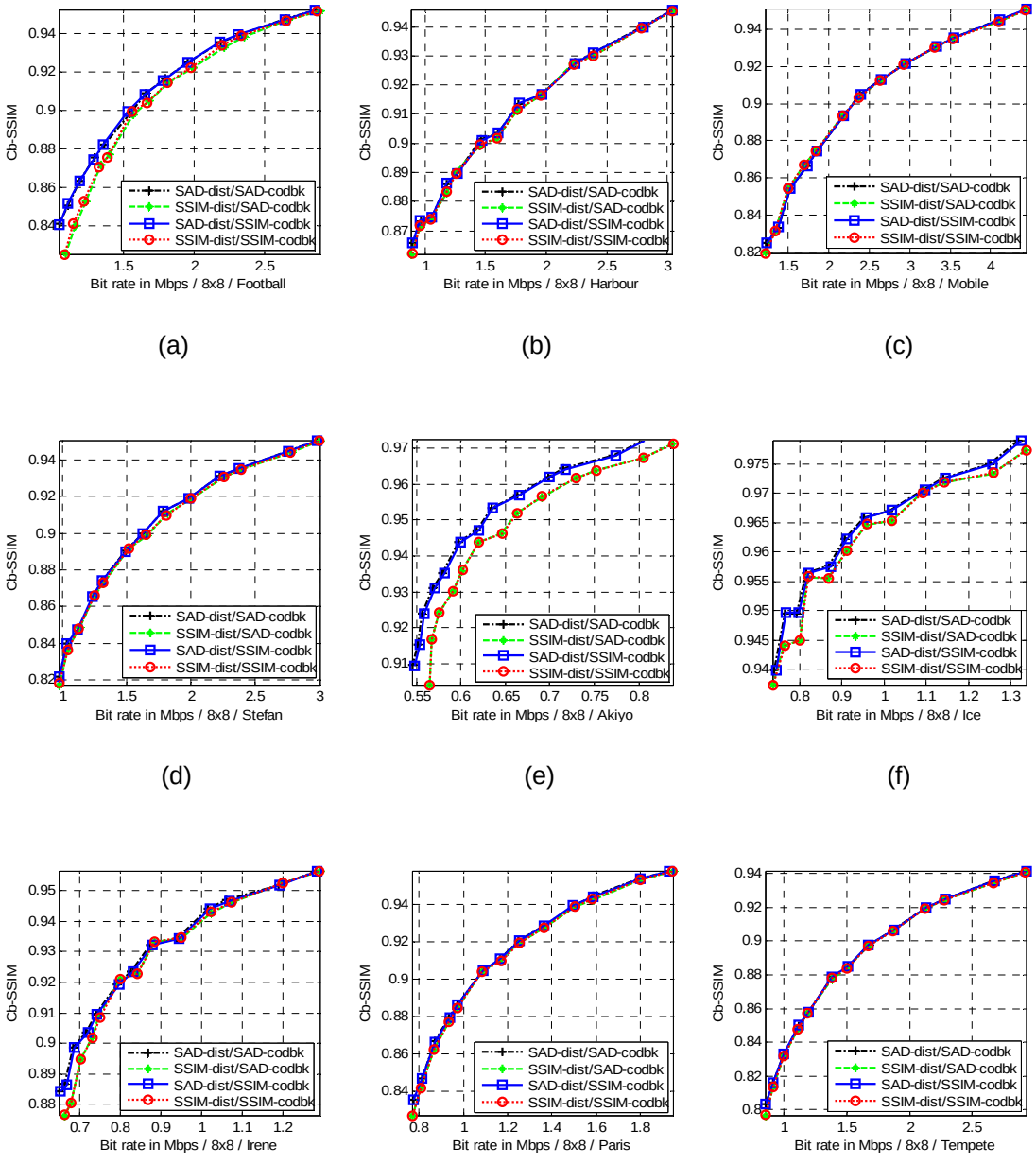


Figure 4.22 Cb-SSIM vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

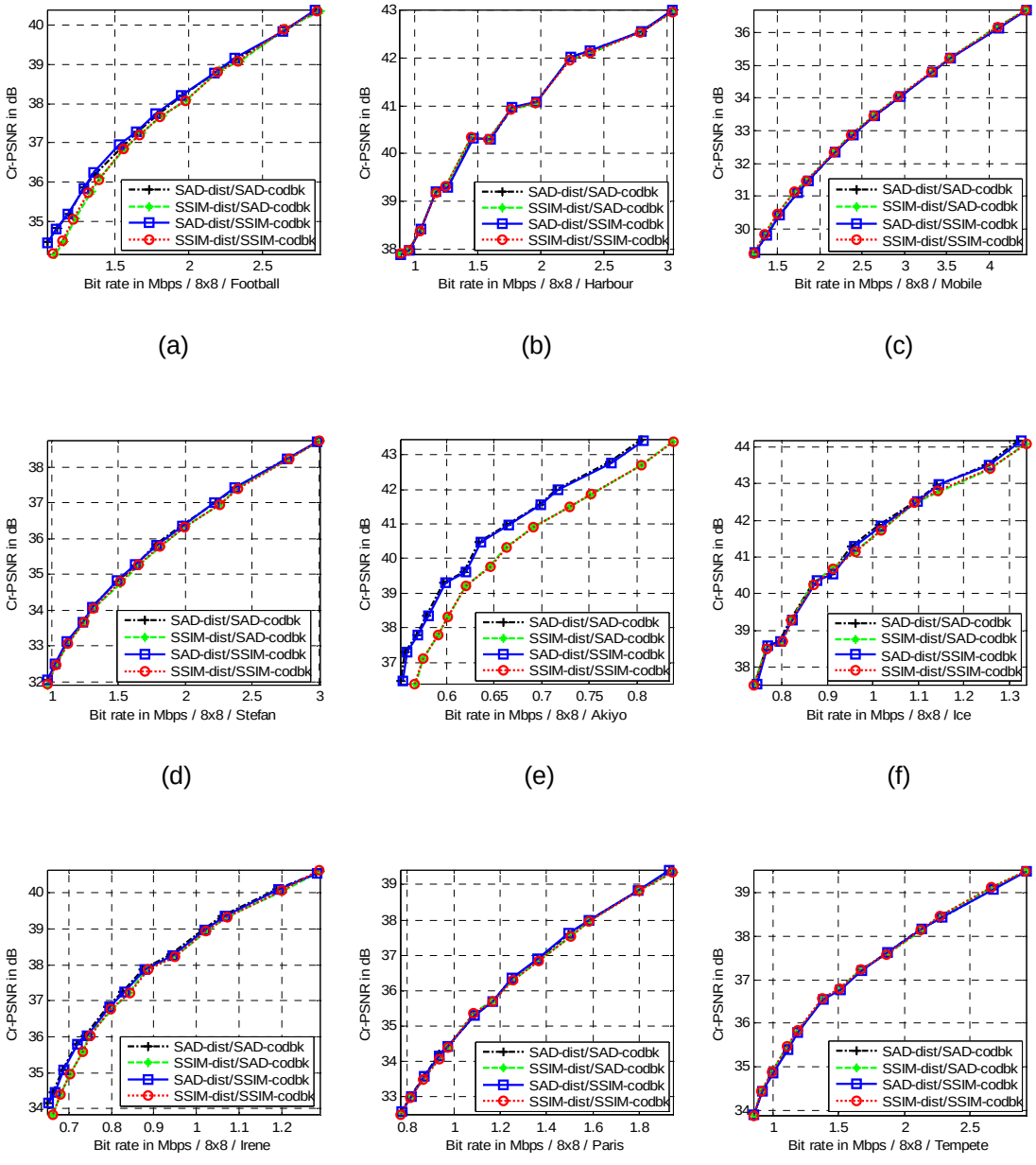


Figure 4.23 Cr-PSNR vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

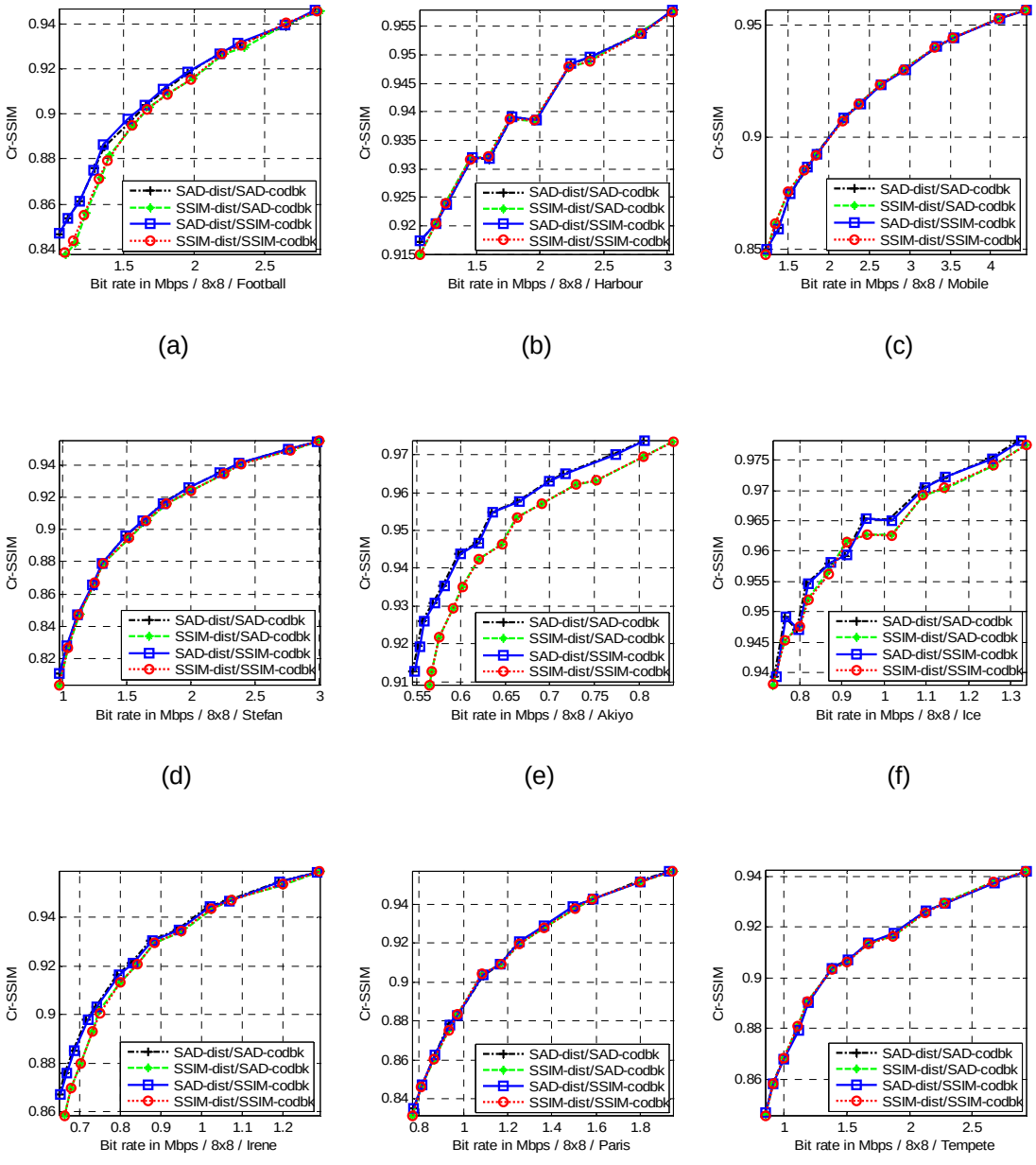


Figure 4.24 Cr-SSIM vs. bit rate comparison, with motion block size (8×8) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

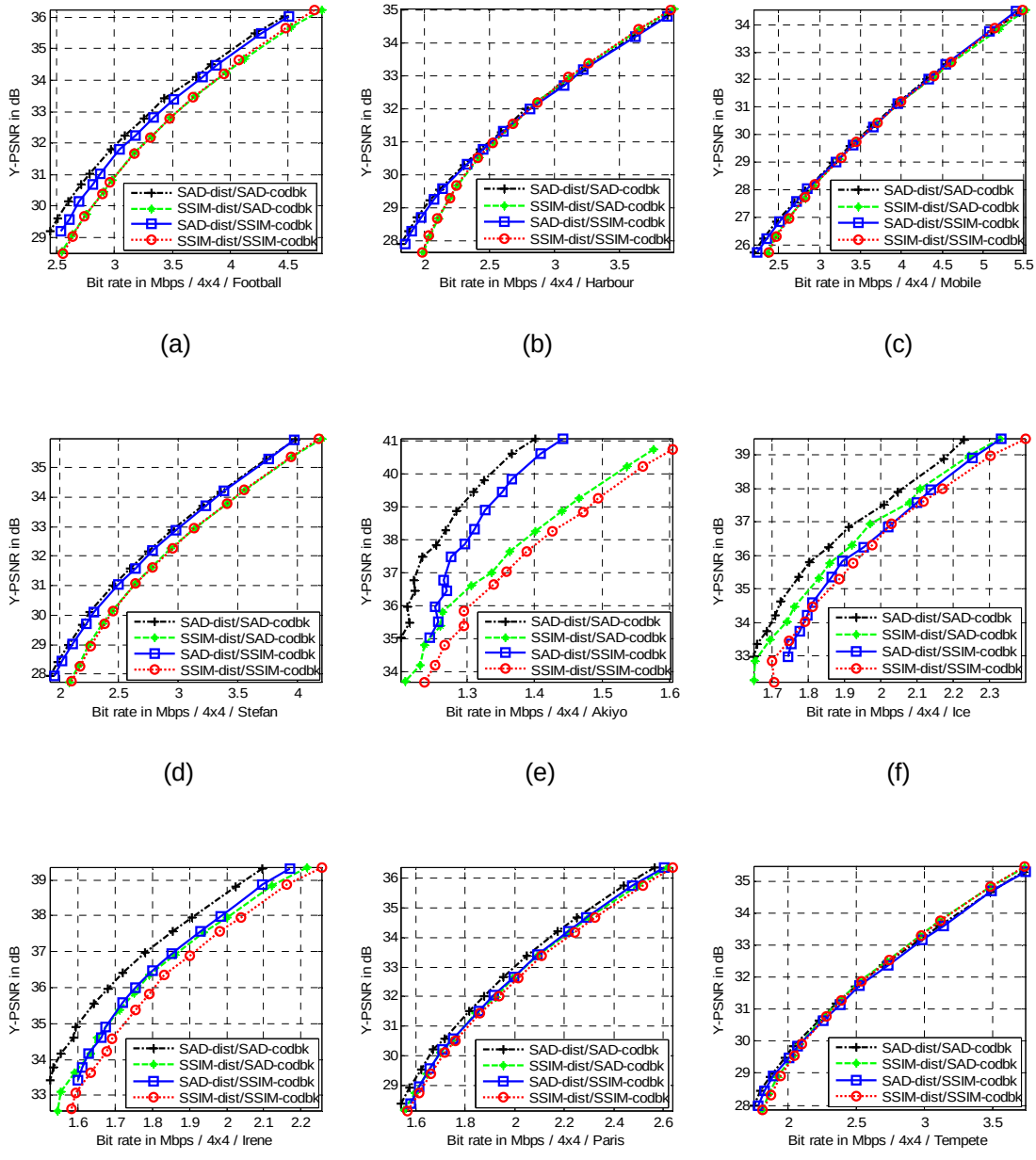


Figure 4.25 Y-PSNR vs. bit rate comparison, with motion block size (4×4) of 9 sequences.
(a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice,
(g) Irene, (h) Paris, and (i) Tempete

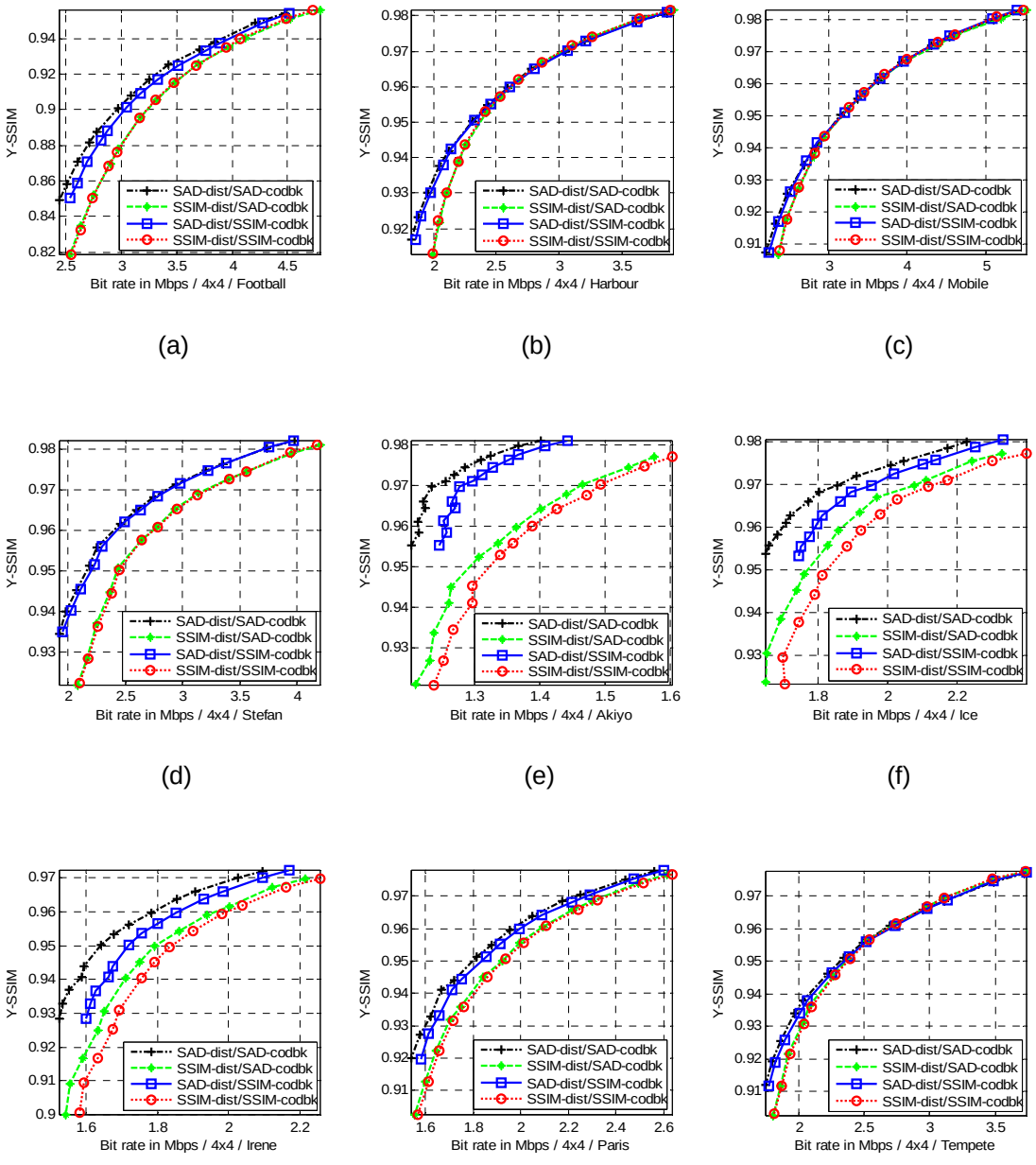


Figure 4.26 Y-SSIM vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

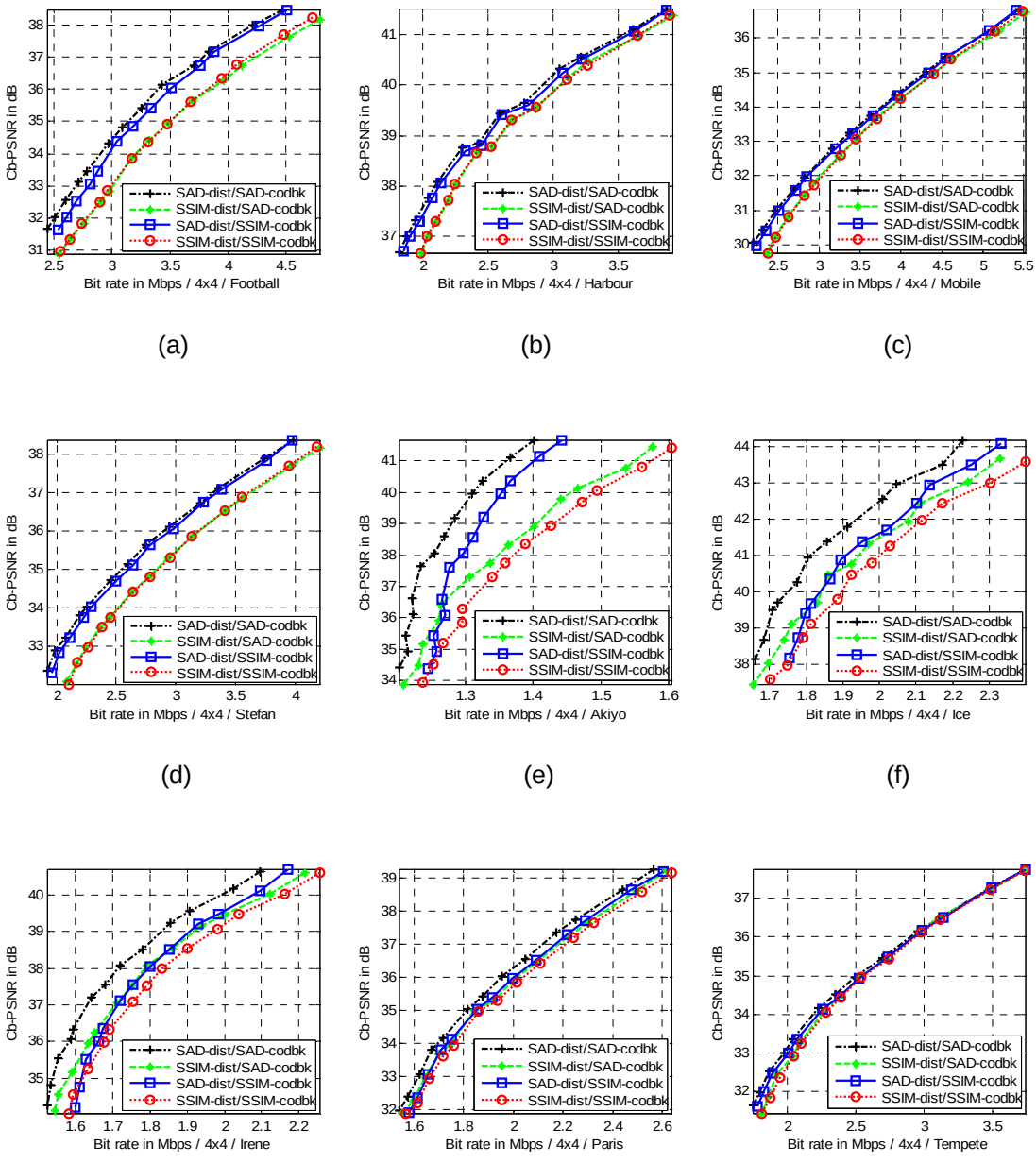


Figure 4.27 Cb-PSNR vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

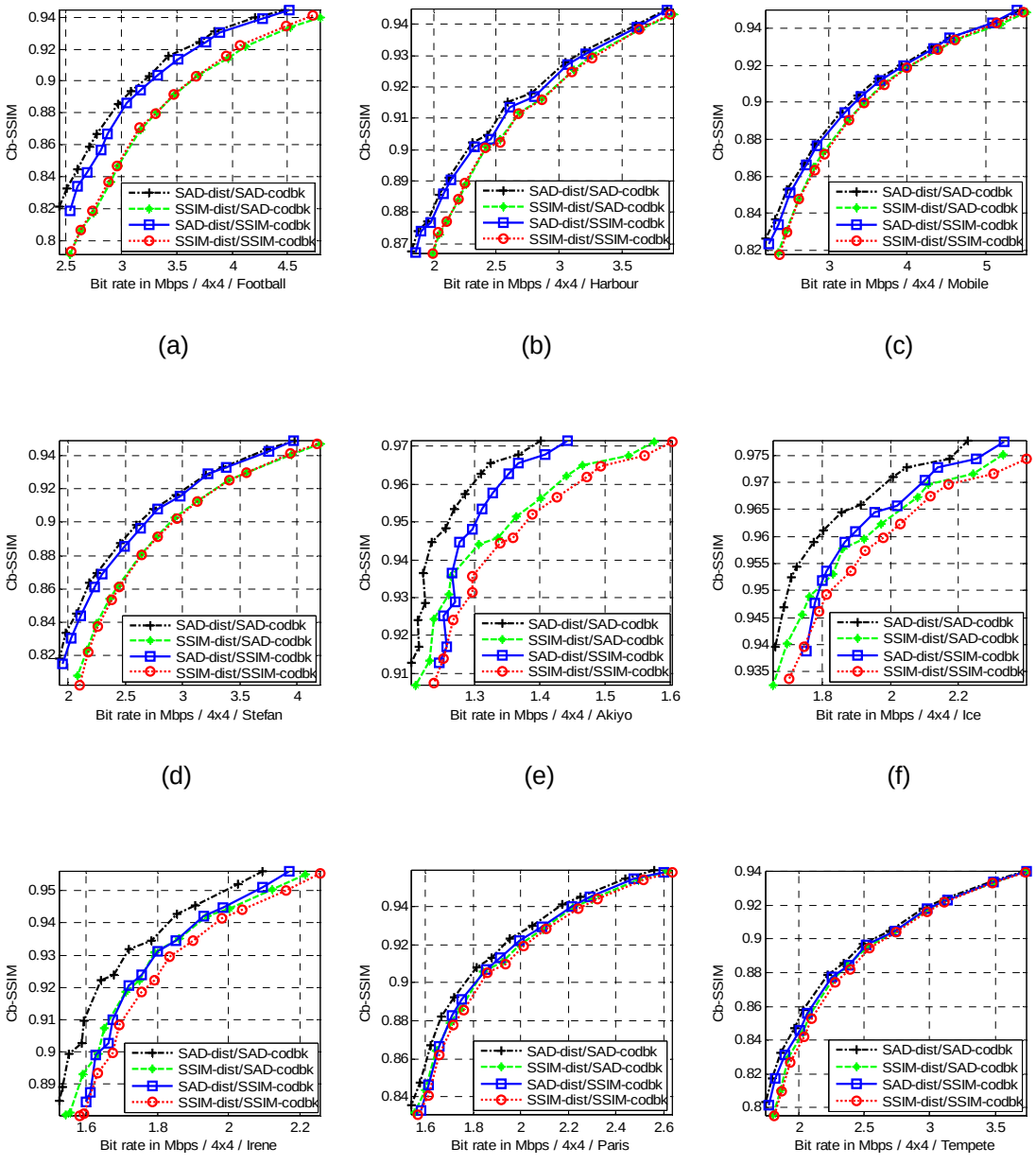


Figure 4.28 Cb-SSIM vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

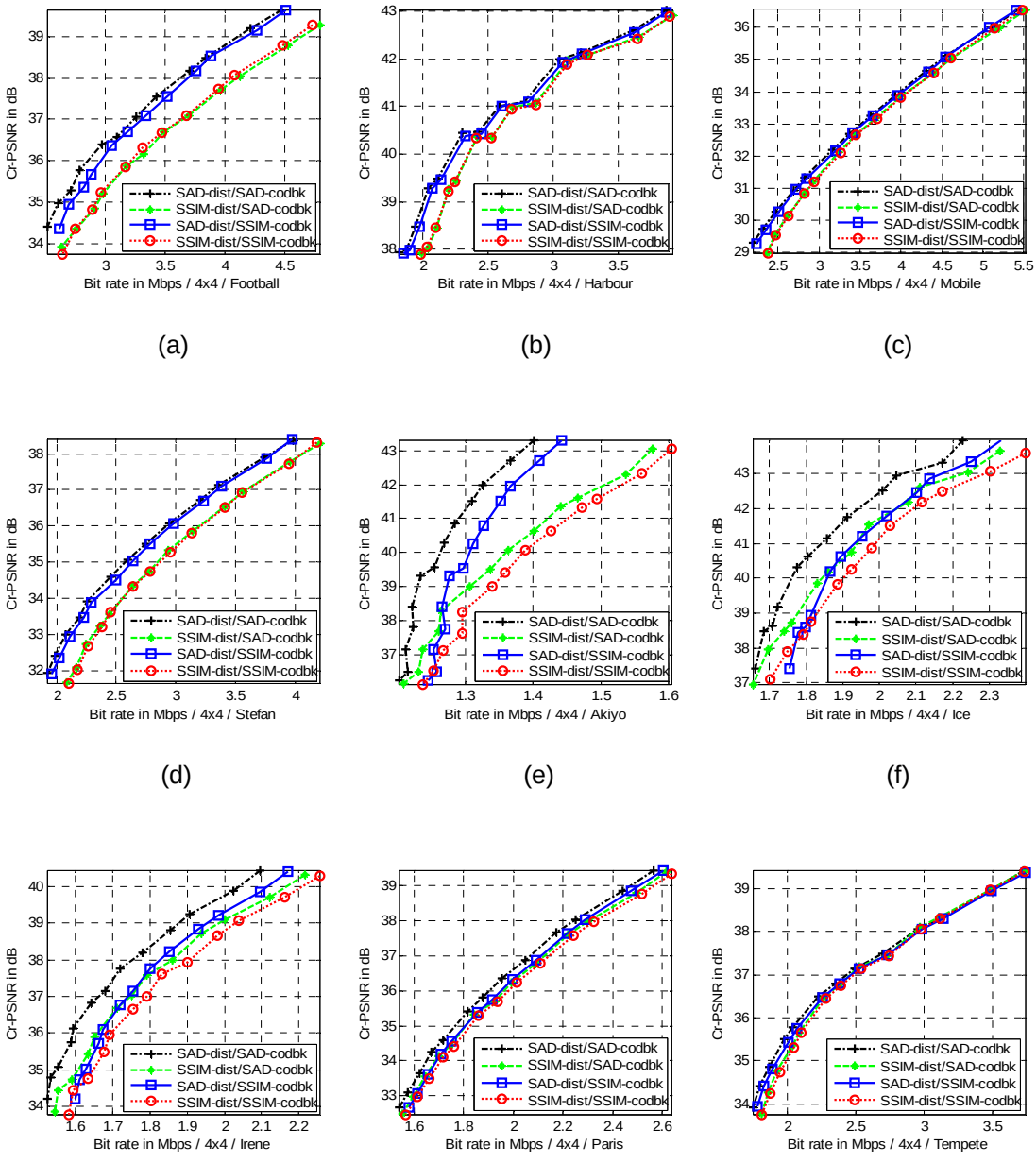


Figure 4.29 Cr-PSNR vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

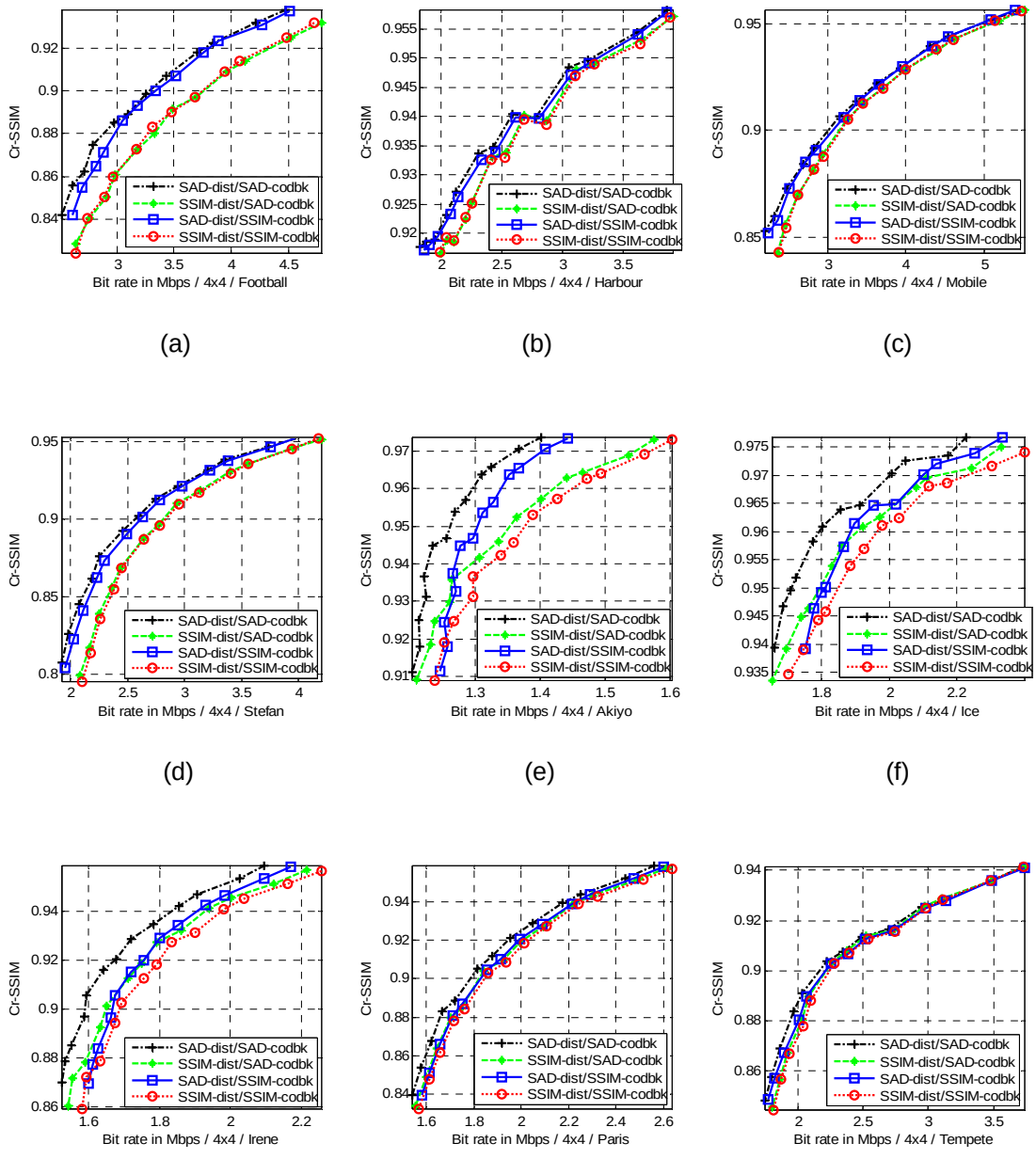


Figure 4.30 Cr-SSIM vs. bit rate comparison, with motion block size (4×4) of 9 sequences. (a) Football, (b) Harbour, (c) Mobile, (d) Stefan, (e) Akiyo, (f) Ice, (g) Irene, (h) Paris, and (i) Tempete

The results show that both PSNR and SSIM provide similar measurements on all motion block sizes and for eight of nine test sequences (except for Ice sequence that gives different measures of PSNR and SSIM).

BD-PSNR [18] (average PSNR differences between two RD curves) is used to compare two RD plots (RGB-PSNR versus bit rate). The results are shown in Tables 4.6–4.7. Table 4.8 summarizes the compared results of using different trained codebooks and using different distortion types in motion prediction. The explanations are as follows.

Consider how both types of trained codebooks affect the rate-distortion performance by comparing the RD results from two types of codebooks with applying the same prediction distortion, for each motion block size. The research considers two cases; case 1, the plots between SAD-based-distortion/SAD-based-codebook and SAD-based-distortion/SSIM-based-codebook (black vs. blue graphs); and case 2, the plots between SSIM-based-distortion/SAD-based-codebook and SSIM-based-distortion/SSIM-based-codebook (green vs. red graphs).

For motion block sizes, (16×16) and (8×8), very low BD-PSNR are shown in Table 4.6. There are no different results of both cases on all nine sequences. For a block size (4×4), both codebook types give no different results in both cases on two of nine sequences (Harbour and Mobile). A SAD-based-codebook gives better results on five of nine sequences (Akiyo, Ice, Irene, Paris, and Tempete).

Next, consider both types of distortions used in motion estimation. This is done by comparing the RD results (BD-PSNR) from two types of prediction distortions by applying the same codebook type to each motion block size, (Table 4.7). This research considers two cases. Case 1 is the plots between SAD-based-distortion and SSIM-based-distortion based on SAD-based-codebook, (black versus green graphs, in Figures 4.7, 4.9, and 4.11). Case 2 is the plots between SAD-based-distortion and SSIM-based-distortion based on SSIM-based-codebook, (blue vs. red graphs, in Figures 4.7, 4.9, and 4.11).

For motion block size (16×16), both types of distortions give no different results for both cases on seven of nine sequences (Harbour, Mobile, Stefan, Ice, Irene, Paris, and Tempete). For the Football sequence, SAD-based-distortion gives 0.19 dB higher BD-PSNR. For the Akiyo sequence, SSIM-based-distortion gives 0.16 dB higher BD-PSNR in Table 4.7. For a motion block size of (8×8), different types of distortions give no different results on the eight of nine sequences, however, for the Akiyo sequence; SAD-based-distortion gives a better result. For block size (4×4), SAD-based-distortion gives better average PSNR compared to SSIM-based-distortion on all sequences (Table 4.7).

Table 4.6 BD-PSNR (dB) Between two RD Curves (RGB-PSNR) with Two Types of Trained Codebooks for Each Motion Block Size. (The notation, SAD-dist/ SAD-codbk, Means to Encode a Test Sequence Using Trained Codebook Based on SAD and Distortion Calculation in Motion Prediction Based on SAD)

Sequence	Block size (16x16)		Block size (8x8)		Block size (4x4)	
	Compare SAD-dist/ SAD-codbk with SAD-dist/ SSIM-codbk	Compare SSIM-dist/ SAD-codbk with SSIM-dist/ SSIM-codbk	Compare SAD-dist/ SAD-codbk with SAD-dist/ SSIM-codbk	Compare SSIM-dist/ SAD-codbk with SSIM-dist/ SSIM-codbk	Compare SAD-dist/ SAD-codbk with SAD-dist/ SSIM-codbk	Compare SSIM-dist/ SAD-codbk with SSIM-dist/ SSIM-codbk
Football	0.0069	0.0080	0.0471	0.0580	-0.2047	0.0373
Harbour	-0.0014	-0.00004	-0.0040	-0.0112	-0.0674	0.0380
Mobile	0.0026	-0.0054	0.0014	-0.0002	-0.0691	-0.0137
Stefan	0.0007	-0.0020	0.0064	0.0130	-0.1013	-0.0207
Akiyo	0.0003	-0.0008	-0.0472	-0.0116	-0.9620	-0.5852
Ice	-0.0067	-0.0054	-0.0669	-0.0085	-0.8883	-0.4482
Irene	-0.0047	-0.0030	-0.0437	0.0161	-0.7118	-0.4038
Paris	-0.0004	0.0018	-0.0109	0.0045	-0.2924	-0.1449
Tempete	0.0032	0.0044	-0.0024	0.0036	-0.8268	-0.5670

Table 4.7 BD-PSNR (dB) Between Two RD Curves (RGB-PSNR) with Two Types of Block Matching Distortions for Each Motion Block Size.

Sequence	Block size (16x16)		Block size (8x8)		Block size (4x4)	
	Compare SAD-dist/ SAD-codbk with SSIM-dist /SAD-codbk	Compare SAD-dist/ SSIM-codbk with SSIM-dist/ SSIM-codbk	Compare SAD-dist/ SAD-codbk with SSIM-dist/ SAD-codbk	Compare SAD-dist/ SSIM-codbk with SSIM-dist/ SSIM-codbk	Compare SAD-dist/ SAD-codbk with SSIM-dist/ SAD-codbk	Compare SAD-dist/ SSIM-codbk with SSIM-dist/ SSIM-codbk
Football	-0.1918	-0.1905	-0.0522	-0.0434	-0.9224	-0.7009
Harbour	-0.0472	-0.0459	0.0684	0.0612	-0.1471	-0.0505
Mobile	-0.0055	-0.0135	0.0518	0.0501	-0.2171	-0.1657
Stefan	-0.0643	-0.0656	-0.0956	-0.0890	-0.7022	-0.6364
Akiyo	0.1589	0.1578	-0.6579	-0.6300	-2.5757	-2.3503
Ice	-0.0987	-0.0975	0.0043	0.0610	-0.7648	-0.3234
Irene	0.0721	0.0735	-0.0763	-0.0163	-0.8037	-0.5003
Paris	-0.0497	-0.0474	-0.0410	-0.0257	-0.3216	-0.1748
Tempete	-0.0285	-0.0272	0.0150	0.0209	-1.8878	-1.1057

Table 4.8 Summary of Result Using Different Trained Codebooks and Distortion Types

Motion block size	For codebooks based on SAD and SSIM with the same prediction distortion	For distortions based on SAD and SSIM with the same trained codebook
(16×16)	No different results on all test sequences	No different results on 7 of 9 sequences (Akiyo sequence, SSIM-based-distortion gives better result. Football sequence, SAD-based-distortion gives better result.)
(8×8)	No different results on all test sequences	No different results on 8 of 9 sequences (Akiyo sequence, SAD-based-distortion gives better result.)
(4×4)	2 of 9 sequences give no different results. 5 of 9 sequences, SAD-based-codebook gives better results.	SAD-based-distortion gives better result on all 9 sequences.

Considering the complexities between SAD and SSIM distortion calculations, the research compares the number of add and multiply operations at block level, based on the distortion equations. The numbers of operations of SAD and SSIM in each block size ($N \times N$) are given in Table 4.9. In Table 4.10, SSIM requires more operations than SAD by 5 times for (16×16), by 5.2 times for (8×8) and by 5.8 times for (4×4).

Table 4.9 Numbers of Operations of SAD and SSIM for Motion Block Size ($N \times N$), Based on (3.4) and (1.8)

Operation	SAD	SSIM
Add	$2N^2 - 1$	$7N^2 + 3$
Multiply	0	$3N^2 + 17$
Total operations per block	$2N^2 - 1$	$10N^2 + 20$

Table 4.10 Total Numbers of Operations Between SAD and SSIM for Three Block Sizes

Size ($N \times N$)	(16×16)	(8×8)	(4×4)
Number of operations on SAD	511	127	31
Number of operations on SSIM	2,580	660	180
Ratio	5	5.2	5.8

CHAPTER 5

SUMMARY AND FUTURE WORK

The proposed video codec takes advantages of a combination of different transform block sizes with a block structure header. The performance is improved over a codec with single transform size. This encoding method can be applied to the H.264 codec to improve its coding efficiency. Since H.264 standard already defines the (4×4) and (8×8) integer transforms, only a new efficient (16×16) integer transform needs to be designed with the appropriate quantization scheme. The decoder requires a few more operations (at most 5 percents) on each macroblock, however, the encoder needs up to 1.83 times more operations compared to H.264 baseline profile.

In the simulation on interframe coding, the codec is implemented and trained with SAD-based and SSIM-based motion predictions. Two sets of codebooks for interframe encoding are created. Then, the codec is tested by different sets of test sequences with motion prediction based on SAD and SSIM. The rate-distortion (RD) results between the two types of distortions are shown. The research proposes SSIM measurement on true color RGB components (RGB-SSIM). The RD results on SSIM measurements do not differ from the results on PSNR measurements. With the same distortion prediction, both types of codebooks show no difference on RD performances for large block sizes (16×16) and (8×8). For a small block size (4×4), SAD-based codebook gives better RD performance. With the same codebook type, both types of distortions in motion prediction have the same performance on a majority of test sequences for block sizes of (16×16) and (8×8). However, for a small block size (4×4), SAD-

distortion based prediction has better performance on the test sequences. However, SSIM is more complex requiring five times the number of operations than SAD.

Further research in improving the coding performance can be described as follows. More spatial prediction modes in intra frame encoding needs to be studied with the encoded bit reduction of the block header. New entropy encoding techniques can be used to reduce the average bit rate. Fast implementation of SSIM by reducing the number of arithmetic operations needs to be explored in order to use SSIM with more efficiency.

APPENDIX A

CONFIGURATION SETTING OF THE REFERENCE SOFTWARES, JM12.0, JPEG2000, AND JPEG

For JM12.0, command line for executing JM12.0 encoder is,

```
lencod –f encoder_baseline.cfg
```

where, `encoder_baseline.cfg` is the configuration text file containing parameters used in the codec.

The parameter configuration for intraframe encoding (III···) is,

```
###############################################################################################
# Files
###############################################################################################
InputFile             = "FOOTBALL_352x288_15_orig_01.yuv"       # Input sequence
InputHeaderLength     = 0     # If the inputfile has a header, state it's length in byte
StartFrame            = 50    # Start frame for encoding. (0-N)
FramesToBeEncoded     = 9     # Number of frames to be coded
FrameRate             = 30.0  # Frame Rate per second (0.1-100.0)
SourceWidth           = 352   # Frame width
SourceHeight          = 288   # Frame height
ReconFile             = "test_rec.yuv"
OutputFile            = "test.264"

###############################################################################################
# Encoder Control
###############################################################################################
ProfileIDC            = 66  # Profile IDC (66=baseline, 77=main, 88=extended)
LevelIDC              = 13  # Level IDC   (e.g. 20 = level 2.0)
IntraPeriod           =  1  # Period of I-Frames (0=only first)
IDRIntraEnable        =  0  # Force IDR Intra  (0=disable 1=enable)
QPISlice              = 42  # Quant. param for I Slices (0-51) ########################
FrameSkip             =  0  # Number of frames to be skipped in input (e.g 2 will code
every third frame)
ChromaQPOffset        =  0  # Chroma QP offset (-51..51)
NumberReferenceFrames = 1 # Number of previous frames used for inter motion search(1-16)

IntraDisableInterOnly  = 0  # Apply Disabling Intra conditions only to Inter Slices
(0:disable/default,1: enable)
Intra4x4ParDisable     = 1  # Disable Vertical & Horizontal 4x4
Intra4x4DiagDisable    = 1  # Disable Diagonal 45degree 4x4
Intra4x4DirDisable     = 1  # Disable Other Diagonal 4x4
Intra16x16ParDisable   = 0  # Disable Vertical & Horizontal 16x16
Intra16x16PlaneDisable = 0  # Disable Planar 16x16
ChromaIntraDisable     = 1  # Disable Intra Chroma modes other than DC
EnableIPCM             = 0  # Enable IPCM macroblock mode

###############################################################################################
# Output Control, NALs
###############################################################################################
SymbolMode             =  0  # Symbol mode (Entropy coding method: 0=UVLC, 1=CABAC)
OutFileMode            =  0  # Output file mode, 0:Annex B, 1:RTP
PartitionMode          =  0  # Partition Mode, 0: no DP, 1: 3 Partitions per Slice

###############################################################################################
# Loop filter parameters
###############################################################################################
LoopFilterDisable     = 1  # Disable loop filter in slice header (0=Filter, 1=No Filter)
```

```
###################################################################################################
# Search Range Restriction / RD Optimization
###################################################################################################
RestrictSearchRange = 0 # restriction for(0: blocks and ref, 1: ref, 2: no restrictions)
RDOptimization      = 0 # rd-optimized mode decision
                        # 0: RD-off (Low complexity mode)
                        # 1: RD-on (High complexity mode)
                      # 2: RD-on (Fast high complexity mode - not work in FREX Profiles)
                      # 3: with losses

###################################################################################################
#Rate control
###################################################################################################
RateControlEnable    =      0   # 0 Disable, 1 Enable

###################################################################################################
#Fast Mode Decision
###################################################################################################
EarlySkipEnable      =      0   # Early skip detection (0: Disable 1: Enable)
SelectiveIntraEnable =      0   # Selective Intra mode decision (0: Disable 1: Enable)

###################################################################################################
#FREXT stuff
###################################################################################################
YUVFormat               = 1     # YUV format (0=4:0:0, 1=4:2:0, 2=4:2:2, 3=4:4:4)
RGBInput                = 0     # 1=RGB input, 0=GBR or YUV input
BitDepthLuma            = 8     # Bit Depth for Luminance (8...12 bits)
BitDepthChroma          = 8     # Bit Depth for Chrominance (8...12 bits)
CbQPOffset              = 0     # Chroma QP offset for Cb-part (-51..51)
CrQPOffset              = 0     # Chroma QP offset for Cr-part (-51..51)
Transform8x8Mode        = 0     # (0: only 4x4 transform, 1: allow using 8x8 transform
additionally, 2: only 8x8 transform)
```

For JPEG2000 reference software (JasPer), command line for encoding is,

```
jasper -f testFbY1.pgm -t pnm -F testFbY1rec034.jpc -T jpc -O rate=0.034
jasper -f testFbCB1.pgm -t pnm -F testFbCB1rec034.jpc -T jpc -O rate=0.034
jasper -f testFbCR1.pgm -t pnm -F testFbCR1rec034.jpc -T jpc -O rate=0.034
```

where, 0.034 is 1/compression rate.
`testFbY1.pgm`, `testFbCB1.pgm`, and `testFbCR1.pgm` are input components frame 1. `testFbY1rec034.jpc`, `testFbCB1rec034.jpc`, and `testFbCR1rec034.jpc` are JPEG2000 encoded files of the corresponding component frame 1.

Command line for decoding a JPEG2000 is,

```
jasper -f testFbY1rec034.jpc -F testFbY1rec034.pgm -T pnm
jasper -f testFbCB1rec034.jpc -F testFbCB1rec034.pgm -T pnm
jasper -f testFbCr1rec034.jpc -F testFbCR1rec034.pgm -T pnm
```

where, `testFbY1rec034.pgm`, `testFbCB1rec034.pgm`, and `testFbCR1rec034.pgm` are the reconstructed component frame 1.

For JPEG reference software (Independent JPEG group), command line for encoding is,

```
cjpeg -quality 45 testFbY1.pgm testFbY1rec45.jpg
cjpeg -quality 45 testFbCB1.pgm testFbCB1rec45.jpg
cjpeg -quality 45 testFbCR1.pgm testFbCR1rec45.jpg
```

where, 45 is a quality parameter.
`testFbY1.pgm`, `testFbCB1.pgm`, and `testFbCR1.pgm` are input components frame 1. `testFbY1rec45.jpg`, `testFbCB1rec45.jpg`, and `testFbCR1rec45.jpg` are JPEG baseline encoded files of the corresponding component frame 1.

APPENDIX B

LIST OF MATLAB FUNCTIONS OF THE PROPOSED CODEC

Table B.1 List of MATLAB Functions for III··· Encoding

Number	Function Name	Description
1	intraEncode	Main function for intraframe encoding
2	qstep_luma_h264	Calculate step size from Quantization Parameter
3	svcPicToBlk	Convert frame into non-overlap blocks in raster order
4	intraDirMode	Perform spatial prediction on I-MB of I-frame
5	interFBMD	Calculate distortion between two blocks
6	svcCodeLook14	Look up codebook table and count number of bits
7	intraAbsDqt	Main function for variable size integer transform encoder and decoder
8	mse_psnr	Calculate MSE and PSNR between two given frames
9	svcDqt	Multi-stage 2x2 Hadamard transform on dc coefficient
10	svcD70	Perform 2-D transform on a 2-D block
11	svcPQR	Create block size selection bits
12	svcH	Generate integer transform matrix with given size
13	svcCLSum16	Sum code-length in each sub-block of a large block
14	svcMuxDqt	Count bit length and obtain a sequence of composite dc coefficients
15	svcMuxDctDqt	Obtain transform of composite block, block size selection bits, and encoded bit-length
16	svcRLC56	Perform run-length encoding of a given block
17	svcDecode	Decode and reconstruct a composite macroblock

Table B.2 List of MATLAB Functions for IPP··· Encoding

Number	Function Name	Description
1	interEncode	Main function for interframe encoding as IPP···
2	qstep_luma_h264	See the corresponding number and function name in Table B.1
3	svcPicToBlk	
4	intraDirMode	
5	interFBMD	
6	svcCodeLook14	
7	intraAbsDqt	
8	mse_psnr	
9	svcDqt	
10	svcD70	
11	svcPQR	
12	svcH	
13	svcCLSum16	
14	svcMuxDqt	
15	svcMuxDctDqt	
16	svcRLC56	
17	svcDecode	
18	interFrameBuff16	Interpolate a reconstructed frame at 1/4- and 1/8- pel
19	interMe20	Perform motion estimation and compensation
20	interMVC	Motion vector calculation (motion prediction)

Table B.3 List of MATLAB Functions for Quality Measurements of a Video Sequence

Number	Function Name	Description
1	mse_psnrRGBfileYUV	Compute RGB-PSNR and MSE from two YUV files
2	mse_psnrYCBCRfileYUV	Compute Y-PSNR, Cb-PSNR, Cr-PSNR from YUVs
3	ssimRGBfileYUV	Compute RGB-SSIM from two YUV files
4	ssimYCBCRfileYUV	Compute Y-SSIM, Cb-SSIM, Cr-SSIM from YUV files

Table B.4 List of MATLAB Functions for Huffman Entropy Codebook Training of I-Frame

Number	Function Name	Description
1	intraHuffCodebookDDqt	Obtain all coefficients for intra codebook training
2	qstep_luma_h264	See the same function name in Table B.1
3	svcH	
4	svcPicToBlk	
5	intraDirMode	
6	interFBMD	
7	svcD70	
8	svcRLC56	
9	svcCode14dic	Create Huffman entropy codebook from coefficients

Table B.5 List of MATLAB Functions for Huffman Entropy Codebook Training of P-Frame

Number	Function Name	Description
1	interHuffCodebook	Obtain all coefficients for inter codebook training
2	qstep_luma_h264	See the same function name in Table B.1
3	svcPicToBlk	
4	intraDirMode	
5	interFBMD	
6	intraAbsDqt	
7	svcDqt	
8	svcD70	
9	svcPQR	
10	svcH	
11	svcCLSum16	
12	svcMuxDqt	
13	svcMuxDctDqt	
14	svcRLC56	
15	svcDecode	
16	interFrameBuff16	See function number 18 in Table B.2
17	interCodebook_Me20	Perform motion estimation and compensation
18	interMVC	See function number 20 in Table B.2
19	interCodebook_intraY	Fixed-size integer transform coding on a luma block
20	interCodebook_intraC	Fixed-size integer transform coding on chroma blocks
21	combineNcount	Combine symbols and counts from two histograms
22	svcCode14dic	See function number 9 in Table B.4

REFERENCES

[1] ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG4-AVC), “Advanced Video Coding for Generic Audiovisual Services,” March 2005. [Online]. Available: http://ecs.itu.ch/cgi-bin/ebookshop

[2] G. Sullivan, T. Wiegand, and H. Yu, *Joint draft 6 of “New profiles for professional applications” amendment to ITUT-Rec*. H.264 & ISO/IEC 14496-10 (Amendment 2 to 2005 edition). in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-V204, Jan. 2007. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2007_01_Marrakech/

[3] S. K. Kwon, A. Tamhankar, and K. R. Rao, “Overview of H.264/MPEG-4 part 10,” *J. Vis. Commun. Image Representation*, vol. 17, pp. 186–216, April 2006.

[4] I.E.G. Richardson, *H.264 and MPEG-4 Video Compression*. West Sussex, UK: Wiley, 2003.

[5] C.-T. Chen, “Adaptive transform coding via quadtree based variable blocksize DCT,” *Proc. IEEE ICASSP*, vol. 3, pp. 1854–1857, 1989.

[6] C. U. Lee. *Adaptive blocksize image compression method and system*. US Patent, Sept. 1995. 5,452,104. [Online]. Available: http://www.uspto.gov

[7] K. Sühring, Ed., (2007) JM 12.0 Reference Software. [Online]. Available: http://iphome.hhi.de/suehring/tml/

[8] A. M. Tourapis, K. Sühring, and G. Sullivan, *Revised H.264/MPEG-4 AVC reference software manual*. in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-Q042, Oct. 2005. [Online]. Avaliable: http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/

[9] M.D. Adams. The JasPer project, University of Victoria, Canada, (2007) JPEG2000 software version 1.900.1 [Online]. Available: http://www.ece.uvic.ca/~mdadams/jasper/

[10] Independent JPEG group, (1998) JPEG software release 6b. [Online]. Available: http://www.ijg.org

[11] H. R. Wu and K. R. Rao, *Digital Video Image Quality and Perceptual Coding*. Boca Raton, FL: CRC Press, 2006.

[12] Z. Wang and A.C. Bovik. *Modern Image Quality Assessment*. San Rafael, CA: Morgan & Claypool, 2006.

[13] Z. Wang *et al*., “Image quality assessment: From error visibility to structural similarity,” *IEEE Trans. Image Processing*, vol. 13, pp. 600–612, Apr. 2004.

[14] Z. Wang. The SSIM Index for Image Quality Assessment, SSIM index MATLAB code. [Online]. Available: http://www.ece.uwaterloo.ca/~z70wang/research/ssim/

[15] Z. Wang, E. P. Simoncelli, and A. C. Bovik, “Multi-scale structural similarity for image quality assessment,” *Asiloma Conference on Signals, Systems, and Computers 2003*, vol. 2, pp. 1398–1402, Nov. 2003.

[16] Z. Wang, L. Lu, and A.C. Bovik, “Video quality assessment based on structural distortion measurement,” *Signal Processing: Image Communication*, vol. 19, pp. 121–132, Feb. 2004.

[17] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, *Digital Image Processing Using MATLAB*. Upper Saddle River, NJ: Pentice-Hall, 2004.

[18] G. Bjontegaard, *Calculation of average PSNR differences between RD-curves*. in STUDY GROUP 16 of ITU-T VCEG, Doc. VCEG-M33, Apr. 2001. [Online]. Available: http://wftp3.itu.ch/av-arch/video-site/0104_Aus/

[19] S. Pateux and J. Jung, *An excel add-in for computing Bjontegaard metric and its evolution.* in STUDY GROUP 16 of ITU-T VCEG, Doc. VCEG-AE07, Jan. 2007. [Online]. Available: http://wftp3.itu.ch/av-arch/video-site/0701_Mar/

[20] M. Wien, “Variable block-size transforms for H.264/AVC”, *IEEE Trans. Circuits Syst. Video Technol*., vol. 13, pp. 604–613, July 2003.

[21] S. Srinivasan et al, “Windows Media Video 9: overview and applications,” *Signal Processing: Image Communication*, vol. 19, pp. 851–875, Oct. 2004.

[22] SMPTE Committee draft: Video Codec VC-1, “Proposed SMPTE Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process,” Jan. 2004. (Not for publication)

[23] W. Gao et al, “AVS – The Chinese next-generation video coding standard,” *NAB 2004*, Las Vegas, April 2004.

[24] Y. Lu *et al*., “Overview of AVS-Video: tools, performance and complexity,” *Proc. SPIE/VCIP 2005*, vol. 5960, pp. 679–690, June 2005.

[25] G. J. Sullivan, P. Topiwala, and A. Luthra, “The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions,” *Proc. SPIE*, vol. 5558, pp. 454–474, Nov. 2004.

[26] Z.-Y. Mai *et al*., “A novel motion estimation method based on structural similarity for H.264 inter prediction,” *IEEE ICASSP 2006*, pp. 913–916, May 2006.

[27] C.-L. Yang, H.-X. Wang, and L.-M. Po, “A novel fast motion estimation algorithm based on SSIM for H.264 video coding,” *Pacific-Rim Conference on Multimedia*, pp. 168–176, Dec. 2007.

[28] H. S. Malvar *et al*., “Low-complexity transform and quantization in H.264/AVC,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 13, pp. 598–603, July 2003.

[29] S. Gordon, D. Marpe, and T. Wiegand, *Simplified use of 8x8 transforms – proposal*. in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-J029, Dec. 2003. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site /2003_12_Waikoloa/

[30] W. K. Cham and Y. T. Chan, "An order-16 integer cosine transform," *IEEE Trans. Signal Processing*, vol. 39, pp. 1205–1208, May 1991.

[31] J. Dong *et al*., "A universal approach to developing fast algorithm for simplified order-16 ICT," *IEEE ISCAS 2007*, pp. 281–284, May 2007.

[32] V. Britanak, P. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms*. Oxford, UK: Academic Press, 2007.

[33] Video Library and Tools, Network System Lab, Simon Fraser University. [Online]. Available: http://nsl.cs.sfu.ca/wiki/index.php/Video_Library_and_Tools

[34] K. Sayood, *Introduction to Data Compression–3rd Edition*. San Francisco: Morgan Kaufmann, 2006.

[35] X. Xu, and Y. He, *Comments on motion estimation algorithms in current JM software*, in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-Q089, Oct. 2005. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2005_10_Nice/

[36] Z. Chen *et al*. *Fast integer pel and fractional pel motion estimation for JVT*. in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-F017, Dec. 2002. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2002_12_Awaji/

[37] X. Yi *et al*. *Improved and simplified fast motion estimation for JM*. in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-P021, July 2005. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2005_07_Poznan/

[38] K.-P. Lim, G. Sullivan, and T. Wiegand, *Text description of joint model reference encoding methods and decoding concealment methods*. in JVT of ISO/IEC MPEG and ITU-T VCEG, Doc. JVT-N046, Jan. 2005. [Online]. Available: http://ftp3.itu.ch/av-arch/jvt-site/2005_01_HongKong/

[39] T. Wedi and H. G. Musmann, "Motion- and aliasing-compensated prediction for hybrid video coding," *IEEE Trans. Circuits Syst. Video Technol*., vol. 13, pp. 577–586, July 2003.

[40] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuits Systs. Video Technol*., vol. 12, pp. 349–355, May 2002.

[41] S. Zhu, and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 287–290, Feb. 2000.

[42] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video, and Audio Coding*. Upper Saddle River, NJ: Prentice Hall PTR, 1996.

BIOGRAPHICAL INFORMATION

Att Kruafak was born in Chiangmai, Thailand, in 1974. He received the B.Eng. Honors degree in telecommunication engineering from King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 1995, and received the Sc.M. degree in electrical engineering from Brown University, Providence, Rhode Island, USA, in 1998. His current research interests include digital image and video coding, processing and communications. He is a member of the Institute of Electrical and Electronics Engineers (IEEE). He has been working in CAT Telecom public company (formerly The Communications Authority of Thailand) as an engineer in international telephone department since 1995. He had been involved in voice circuit provisioning, ITU-T Signaling number 7 planning of the international gateways.